

Microsoft Fabric lifecycle management documentation

Microsoft Fabric's lifecycle management tools enable efficient product development, continuous updates, fast releases, and ongoing feature enhancements.

About Lifecycle management

OVERVIEW

[What is CI/CD Lifecycle management?](#)

[What is Git integration?](#)

[What is deployment pipelines?](#)

[What is a Variable library?](#)

Get started

QUICKSTART

[Get started with Git integration](#)

[Get started with deployment pipelines](#)

[Get started with Variable libraries](#)

CONCEPT

[Git integration concepts](#)

[The deployment pipelines process](#)

[Variable library variable types](#)

REFERENCE

[Frequently asked questions about the Fabric lifecycle management tools](#)

[Troubleshoot lifecycle management issues](#)

What's new in CI/CD?

WHAT'S NEW

[Variable libraries](#)

[Deployment pipelines new interface](#)

[Supported items](#)

How to guides

HOW-TO GUIDE

[Manage Git branches](#)

[Resolve conflicts in Git](#)

[Lifecycle management best practices](#)

TUTORIAL

[Lifecycle management tutorial](#)

Automate lifecycle management workflows using APIs

HOW-TO GUIDE

[Automate Git workflows](#)

[Automate deployment pipelines](#)

[Automate Variable libraries](#)

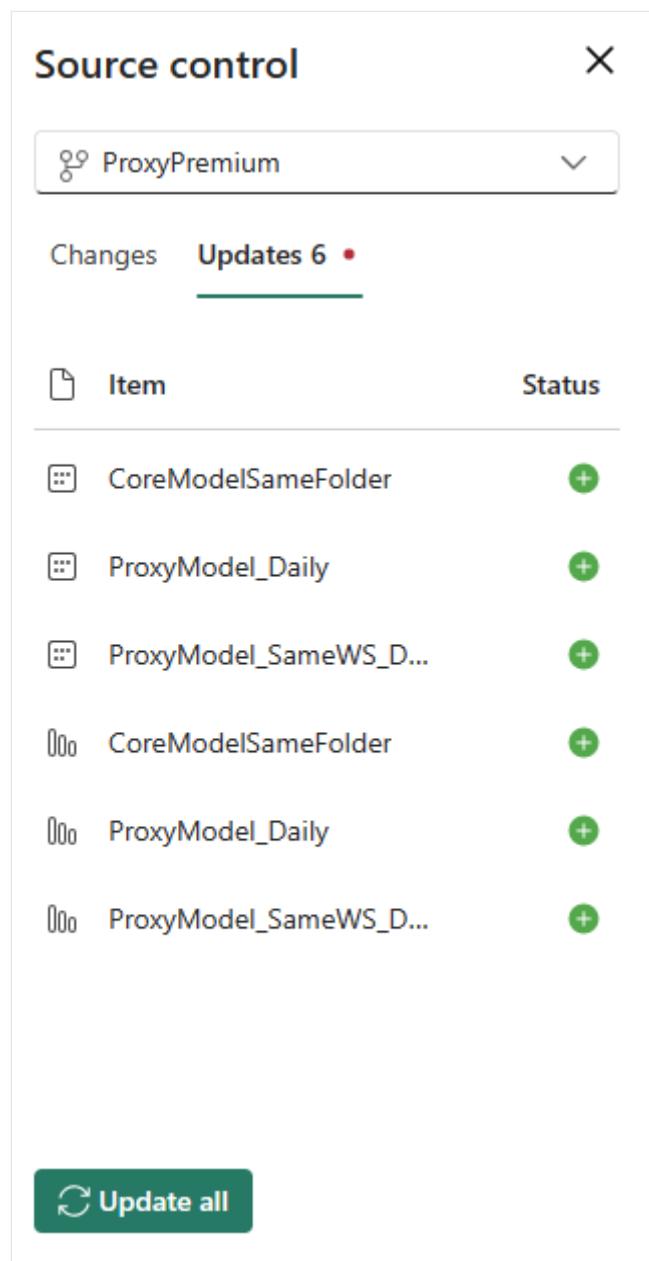
REFERENCE

[Fabric REST APIs](#)

What is lifecycle management in Microsoft Fabric?

Microsoft Fabric's lifecycle management tools provide a standardized system for communication and collaboration between all members of the development team throughout the life of the product. Lifecycle management facilitates an effective process for releasing products quickly by continuously delivering updated content into production and ensuring an ongoing flow of new features and bug fixes using the most efficient delivery method. There are two main components of lifecycle management in Fabric:

Git integration



The screenshot shows the Microsoft Fabric Source control interface. At the top, it displays the title "Source control" and a dropdown menu set to "ProxyPremium". Below this, there are two tabs: "Changes" and "Updates 6 •", with "Updates" being the active tab. The main area lists six items, each with a status indicator (green plus sign) and a "Details" link. The items are:

Item	Status
CoreModelSameFolder	+
ProxyModel_Daily	+
ProxyModel_SameWS_D...	+
CoreModelSameFolder	+
ProxyModel_Daily	+
ProxyModel_SameWS_D...	+

At the bottom left, there is a green button labeled "Update all" with a circular arrow icon.

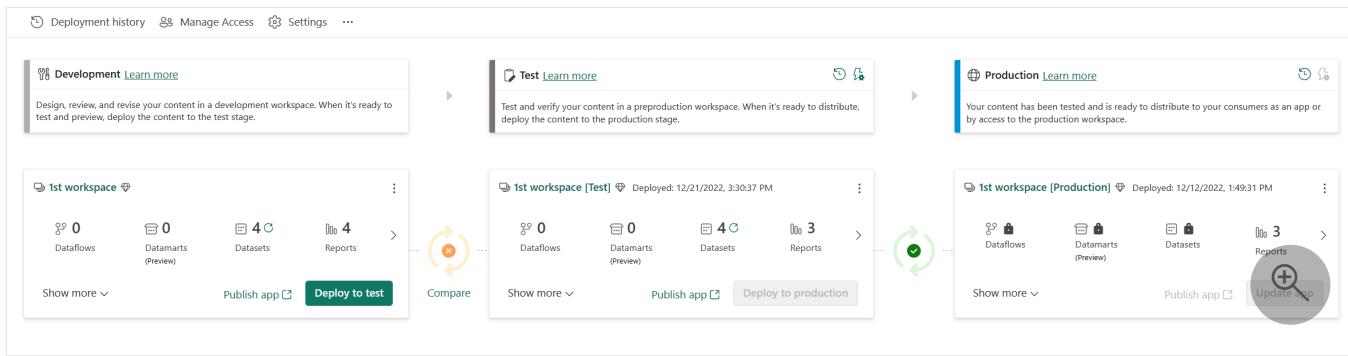
With Fabric's [Git integration](#) process, incremental workspace updates can be made frequently and reliably by multiple developers. By applying Git advantages and best practices, developers

can collaborate and ensure that content changes get to the workspace quickly and reliably. When ready, the delivery process can then deliver the content to deployment pipelines for testing and distribution.

⚠ Note

Some of the items for CI/CD are in preview. See the list of supported item for the [Git integration](#) and [deployment pipeline](#) features.

Delivery through deployment pipelines



Fabric's [deployment pipelines](#) automates the [delivery](#) of modified content to environments like testing and production. It allows teams to produce updates in short cycles with high speed, frequency, and reliability. Content can be released at any time with a simple, repeatable deployment process.

For the most efficient lifecycle management experience in Fabric, connect your developer workspace to Git, and deploy from the connected workspace using deployment pipelines.

Variable library

With [Variable libraries](#), customers can:

- Define and manage variables (user-defined variables) in a unified way for all workspace items.
- Use the variables in different places in the product: In item definitions (such as queries), as reference to other items (Lakehouse ID), and more.
- Reuse variables across fabric workloads and items (for example, several items in the workspace can refer to the same variable).
- CI/CD - Use variables to adjust values based on the release pipeline stage.

Related content

- End to end lifecycle management tutorial
 - Deployment pipelines
 - Git integration
-

Last updated on 12/15/2025

Tutorial: Lifecycle management in Fabric

In this tutorial, you go through the whole process of loading data into your workspace, and using deployment pipelines together with Git integration to collaborate with others in the development, testing, and publication of your data and reports.

! Note

Some Git integration items are in preview. For more information, see the list of [supported items](#).

Prerequisites

To integrate Git with your Microsoft Fabric workspace, you need to set up the following prerequisites for both Fabric and Git.

Fabric prerequisites

To access the Git integration feature, you need a [Fabric capacity](#). A Fabric capacity is required to use all supported Fabric items. If you don't have one yet, [sign up for a free trial](#). Customers that already have a [Power BI Premium capacity](#), can use that capacity, but keep in mind that [certain Power BI SKUs only support Power BI items](#).

In addition, the following [tenant switches](#) must be enabled from the Admin portal:

- [Users can create Fabric items](#)
- [Users can synchronize workspace items with their Git repositories](#)
- [Create workspaces](#) (only if you want to branch out to a new workspace.)
- [Users can synchronize workspace items with GitHub repositories](#): For GitHub users only

These switches can be enabled by the tenant admin, capacity admin, or workspace admin, depending on your [organization's settings](#).

Git prerequisites

Git integration is currently supported for Azure DevOps and GitHub. To use Git integration with your Fabric workspace, you need the following in either Azure DevOps or GitHub:

Azure DevOps

- An Active **Azure DevOps account** registered to same Fabric user (supported even if Azure DevOps organization reside in a different tenant than Fabric tenant). [Create a free account ↗](#).
- Access to an existing repository.

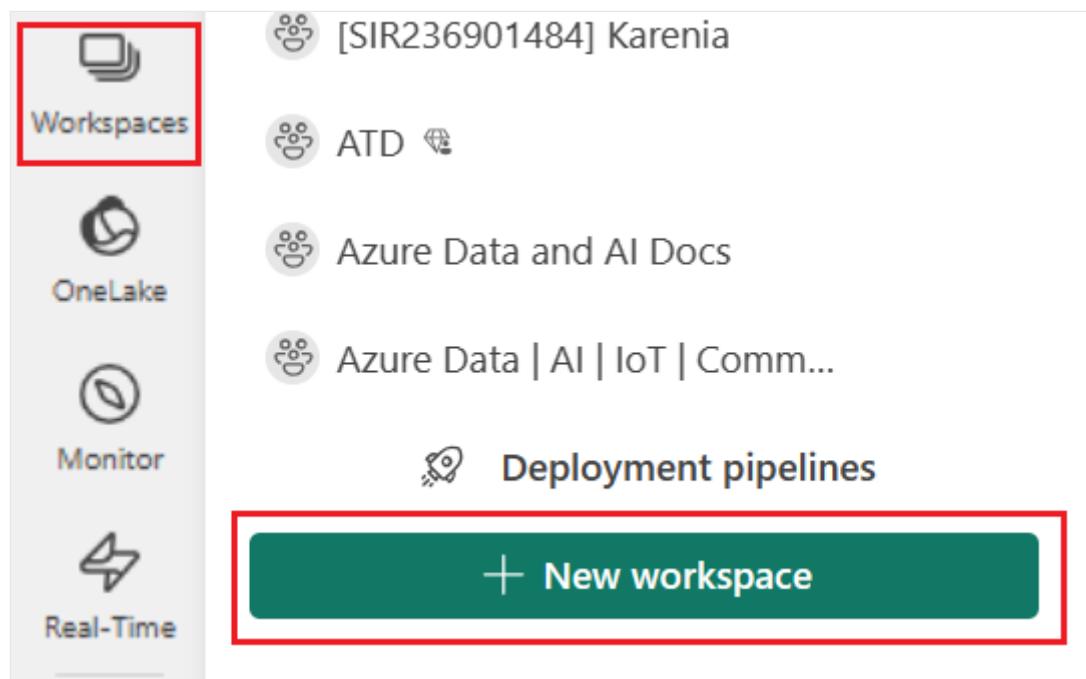
- Download the [FoodSales.pbix ↗](#) file into a Git repo that you can edit. We use this sample file in this tutorial. Alternatively, you can use your own semantic model and report, if you prefer.

If you already have admin rights to a workspace with data, you can skip to [step 3](#).

Step 1: Create a Premium workspace

To create a new workspace and assign it a license:

1. From the left navigation bar of the **Power BI** experience, select **Workspaces > + New workspace**.



2. Name the workspace **FoodSalesWS**.
3. (Optional) Add a description.

Create a workspace

Name *

FoodSalesWS

Available

Description

Sample workspace for application lifecycle management tutorial

Domain (preview) ⓘ

Assign to a domain (optional)



[Learn more about workspace settings](#) ↗

Workspace image



Upload

Reset

Advanced ▾

4. Expand the **Advanced** section to reveal **License mode**.

5. Select either **Trial** or **Premium capacity**.

Advanced ^

Contact list * ⓘ



me (Owner) X

Enter users and groups

License mode ⓘ

Pro

Select Pro to use basic Power BI features and collaborate on reports, dashboards, and scorecards. To access a Pro workspace, users need Pro per-user licenses. [Learn more ↗](#)

Trial

Select the free trial per-user license to try all the new features and experiences in Microsoft Fabric for 60 days. A Microsoft Fabric trial license allows users to create Microsoft Fabric items and collaborate with others in a Microsoft Fabric trial capacity. Explore new capabilities in Power BI, Data Factory, Data Engineering, and Real-Time Analytics, among others. [Learn more ↗](#)

Premium per-user

Select Premium per-user to collaborate using Power BI Premium features, including paginated reports, dataflows, and datamarts. To collaborate and share content in a Premium per-user workspace, users need Premium per-user licenses. [Learn more ↗](#)

Premium capacity

Select premium capacity if the workspace will be hosted in a premium capacity. When you share, collaborate on, and distribute Power BI and Microsoft Fabric content, users in the viewer role can access this content without needing a Pro or Premium per-user license. [Learn more ↗](#)

Embedded ⓘ

Select embedded if the workspace will be hosted in an Azure embedded capacity. ISVs and developers use Power BI Embedded to embed visuals and analytics in their applications. [Learn more ↗](#)

Fabric capacity

Select Fabric capacity if the workspace will be hosted in a Microsoft Fabric capacity. With Fabric canaries, users can create Microsoft Fabric items and collaborate with others using Fabric.

Apply

Cancel

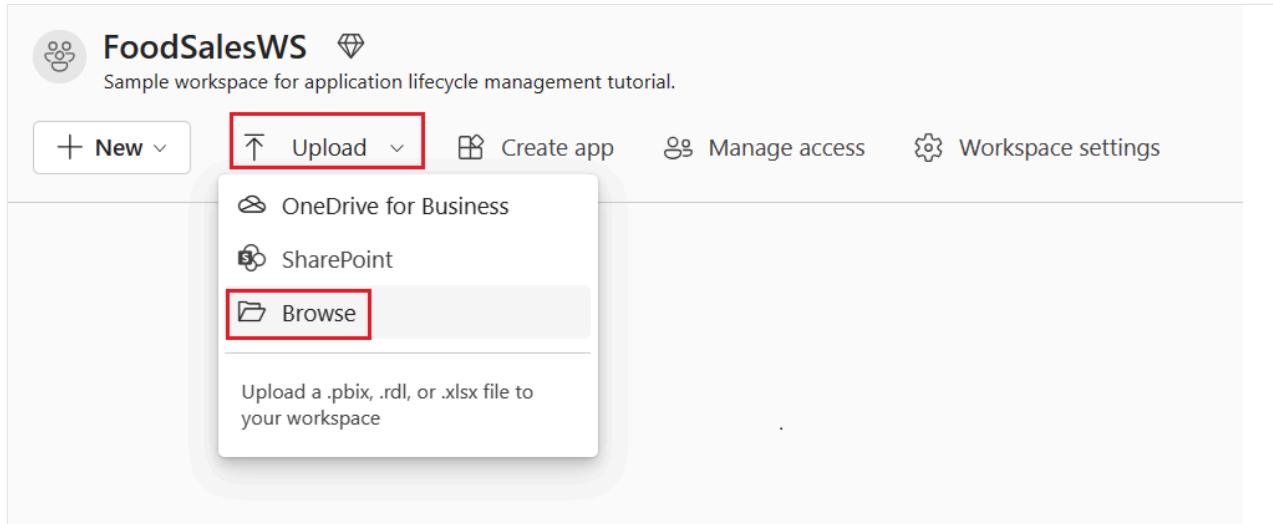
6. Select **Apply**.

For more on creating a workspace, see [Create a workspace](#).

Step 2: Load content into the workspace

You can upload content from OneDrive, SharePoint, or a local file. In this tutorial, we load a **.pbix** file.

1. From the top menu bar, select **Upload > Browse**.



2. Browse to the location of the **FoodSales.pbix** file you [downloaded earlier](#), or load your own sample semantic model and report.

You now have a workspace with content in it for you and your team to work on.

Name	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
FoodSales	Report	FoodSalesWS	26/09/23, 17:50:48	—	—	Non-Business	<input checked="" type="checkbox"/> No
FoodSales	Semantic model	FoodSalesWS	26/09/23, 17:50:48	N/A	—	Non-Business	<input type="checkbox"/> Yes
FoodSales.pbix	Dashboard	FoodSalesWS	—	—	—	Confidential\Micro...	<input checked="" type="checkbox"/> No

Edit credentials - first time only

Before you create a deployment pipeline, you need to set the credentials. This step only needs to be done once for each semantic model. After your credentials are set for this semantic model, you won't have to set them again.

1. Go to **Settings > Power BI settings**.

The screenshot shows the 'Settings' screen of the Power BI mobile application. At the top, there is a navigation bar with icons for notifications, settings (highlighted with a red box), download, help, and profile. Below the navigation bar, the title 'Settings' is displayed. The main content area is organized into sections:

- Preferences**
 - [General →](#)
 - [Notifications →](#)
 - [Item settings →](#)
 - [Developer settings →](#)
-
- Resources and extensions**
 - [Manage group storage →](#)
 - [Power BI settings →](#) (This item is highlighted with a red box)
 - [Manage connections and gateways →](#)
 - [Manage embed codes →](#)
 - [Azure Analysis Services migrations →](#)
-
- Governance and insights**
 - [Admin portal →](#)

2. Select Semantic models > Data source credentials > Edit credentials.

Settings for FoodSales

[View dataset](#)

This dataset has been configured by [monaberdugo@microsoft.com](#).

[Refresh history](#)

[Dataset description](#)

Describe the contents of this dataset.

500 characters left

[Apply](#) [Discard](#)

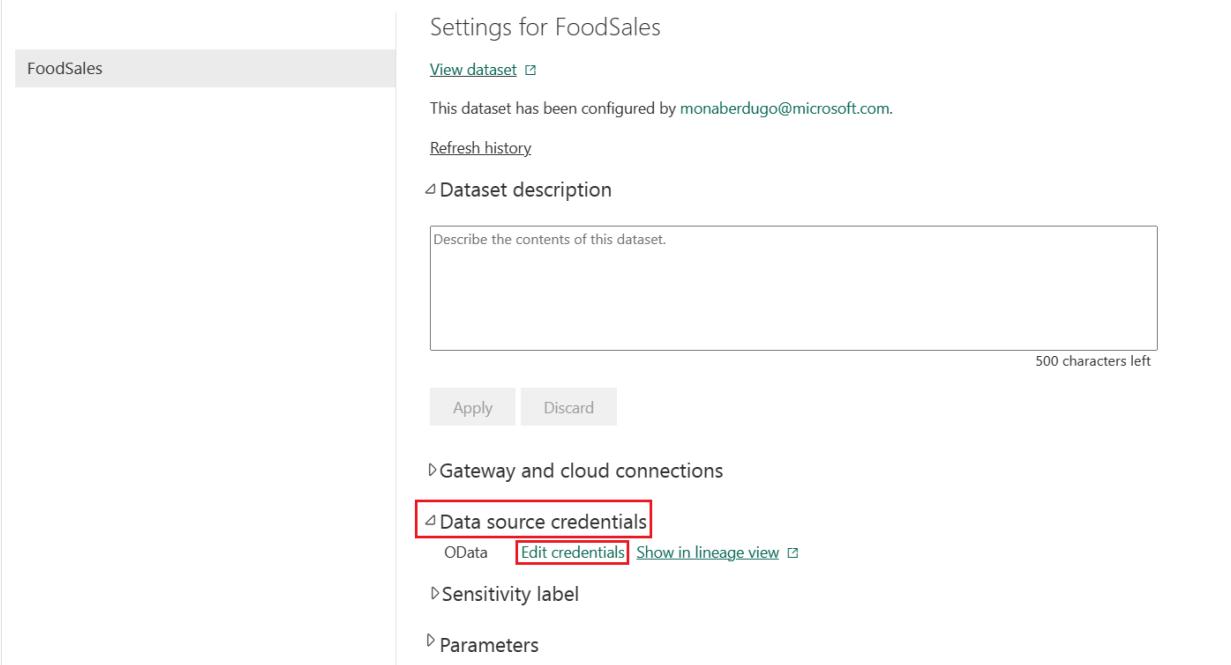
[Gateway and cloud connections](#)

[Data source credentials](#)

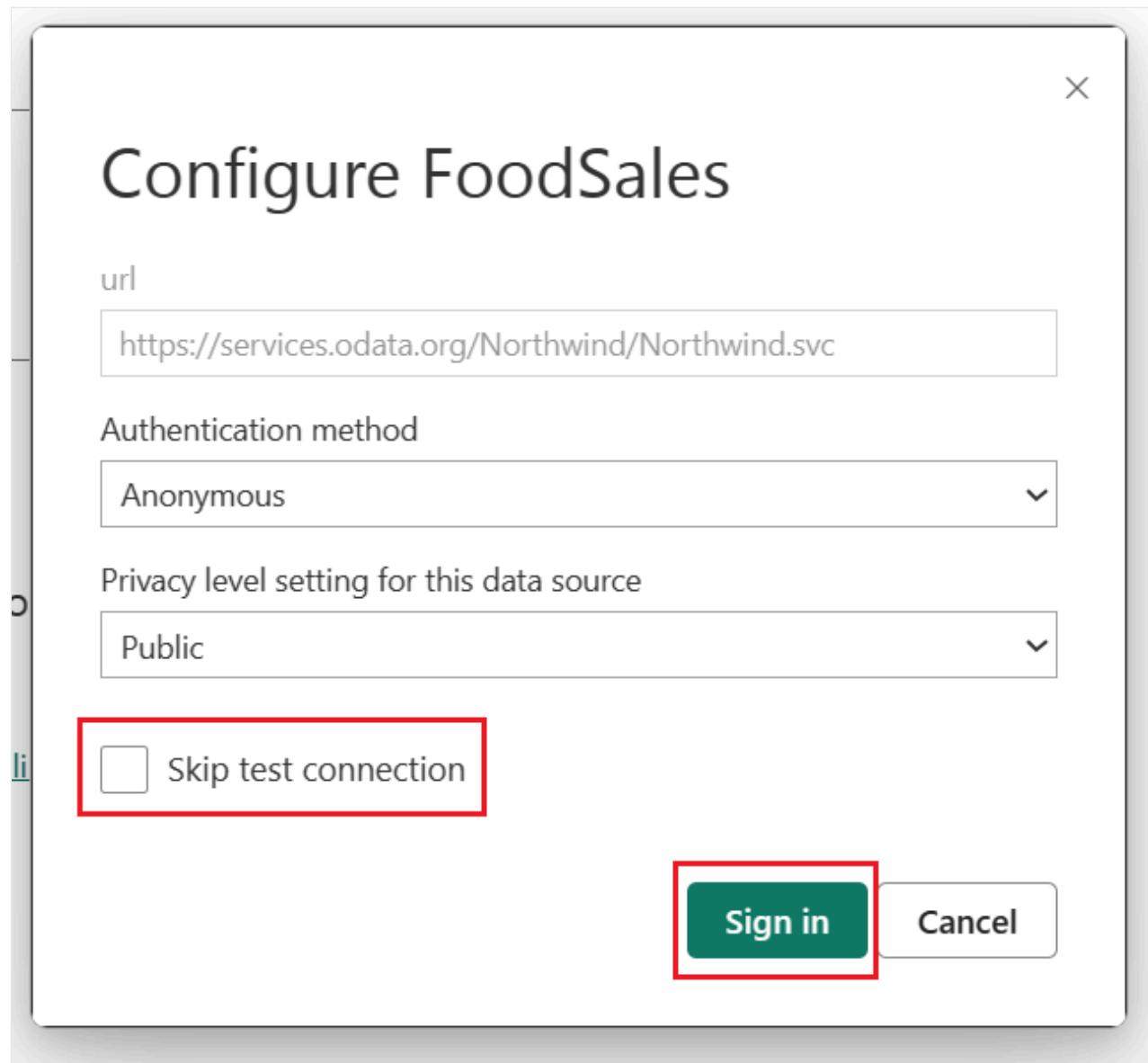
OData [Edit credentials](#) [Show in lineage view](#)

[Sensitivity label](#)

[Parameters](#)



3. Set the **Authentication** method to *Anonymous*, the **Privacy level** to *Public*, and uncheck the **Skip test connection** box.



4. Select **Sign in**. The connection is tested and credentials set.

You can now create a deployment pipeline.

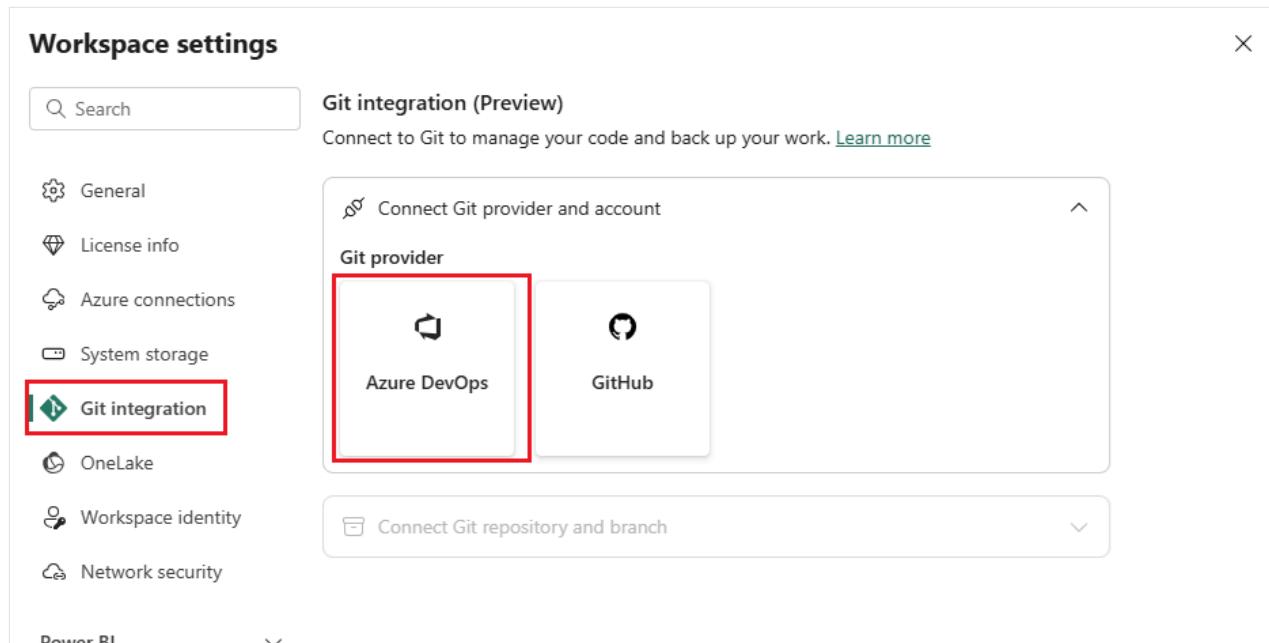
Step 3: Connect the team's development workspace to git

The entire team shares this workspace and each member of the team can edit it. By connecting this workspace to git, you can keep track of all the changes and revert back to previous versions if necessary. When all the changes are merged into this shared branch, deploy this workspace to production using the deployment pipeline.

Read more about version control with Git in [Introduction to Git integration](#).

Let's connect this workspace to the main branch of your Git repo so all team members can edit it and create pull requests. Follow these steps if you're using an Azure DevOps repo. If you're using a GitHub repo, follow the directions in [Connect a workspace to a GitHub repo](#).

1. Go to **Workspace settings** in the top right corner.
2. Select **Git integration**.
3. Select **Azure DevOps**. You're automatically signed into the Azure Repos account registered to the Microsoft Entra user signed into the workspace.



4. From the dropdown menu, specify the following details about the branch you want to connect to:
 - Organization
 - Project
 - Git repository
 - Select *main* (or *master*) branch
 - Type the name of folder in the repo where the *.pbix* file located. This folder will be synced with the workspace.

Workspace settings

Search

Git integration

Connect to Git with Azure DevOps to manage your code and back up your work. [Learn more](#)

About

Premium

Azure connections

System storage

Git integration

Other

Power BI

Data
Engineering/Science

View Azure DevOps account

Connect Git repository and branch

Organization *

MyOrg

Project *

TeamProject

Git repository * ⓘ

Sample-Workload

Branch * ⓘ

main

Git folder ⓘ

ProjectFolder

Connect and sync

Cancel

5. Select Connect and sync.

After you connect, the Workspace displays information about source control that allows you to view the connected branch, the status of each item in the branch and the time of the last sync. The Source control icon shows 0 because the items in the workspace Git repo are identical.

The screenshot shows the Azure Data Studio interface for a workspace named "FoodSalesWS". The top navigation bar includes "New", "Upload", "Create deployment pipeline", "Create app", "Manage access", and "Source control". The "Source control" button is highlighted with a red box. Below the navigation is a search bar and filter options. The main area displays a table of workspace items:

Name	Git status	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
"FoodSales"	Synced	Report	FoodSalesWS	25/07/23, 12:48:56	—	—	Non-Business	No
"FoodSales"	Synced	Dataset	FoodSalesWS	25/07/23, 12:48:56	N/A	—	Non-Business	No
FoodSales.pbix	Unsupported	Dashboard	FoodSalesWS	—	—	—	—	No

At the bottom, a footer bar shows "main" and "Last synced: 7/25/2023 at 1:10 PM abc123abc".

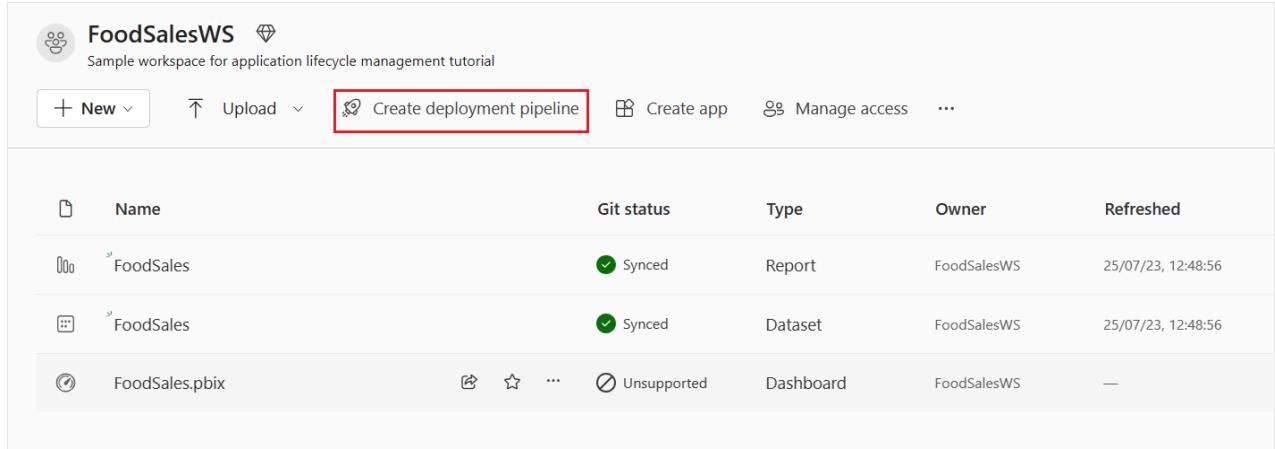
Now the workspace is synced with the main branch of your Git repo making it easy to keep track of changes.

For more information about connecting to git, see [Connect a workspace to an Azure repo](#).

Step 4: Create a deployment pipeline

In order to share this workspace with others and use it for various stages of testing and development, we need to create a deployment pipeline. You can read about how deployment pipelines work in [Introduction to deployment pipelines](#). To create a deployment pipeline and assign the workspace to the development stage, do the following steps:

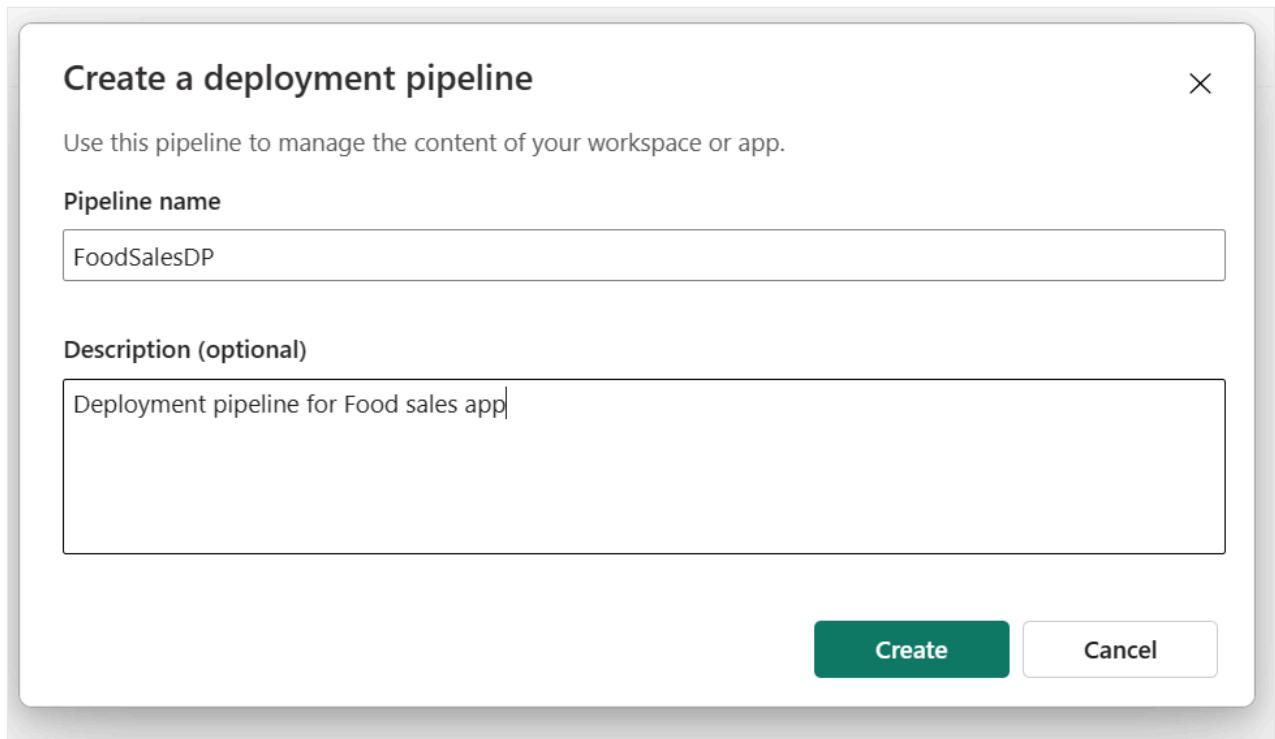
1. From the workspace home page, select **Create deployment pipeline**.



The screenshot shows the 'FoodSalesWS' workspace home page. At the top, there's a navigation bar with 'New', 'Upload', 'Create deployment pipeline' (which is highlighted with a red border), 'Create app', 'Manage access', and an ellipsis. Below the navigation is a table listing workspace items:

	Name	Git status	Type	Owner	Refreshed
Report	FoodSales	Synced	Report	FoodSalesWS	25/07/23, 12:48:56
Dataset	FoodSales	Synced	Dataset	FoodSalesWS	25/07/23, 12:48:56
Dashboard	FoodSales.pbix	Unsupported	Dashboard	FoodSalesWS	—

2. Name your pipeline *FoodSalesDP*, give it a description (optional) and select **Next**.



The screenshot shows the 'Create a deployment pipeline' dialog box. It includes fields for 'Pipeline name' (set to 'FoodSalesDP') and 'Description (optional)' (set to 'Deployment pipeline for Food sales app'). At the bottom are 'Create' and 'Cancel' buttons.

3. Accept the default three stages to your pipeline, and select **Create**.

Customize your stages

X

Define the stages you want to be included in your pipeline and give each stage a name.

+ Add

1 Development



2 Test



3 Production



Create

Cancel

4. Assign the FoodSalesWS workspace to the Development stage.

Assign your workspace to a stage

X

Choose a stage to assign 'FoodSalesWS' to or skip for now and assign anytime from the pipelines page.

Development

Test

Production

Assign

Skip

The development stage of the deployment pipeline shows one semantic model, one report, and one dashboard. The other stages are empty.

The screenshot shows the Microsoft Power BI Deployment Pipeline interface. At the top, there's a header with the pipeline name "FoodSalesDP" and a subtitle "Deployment pipeline for Food sales app". Below the header are navigation links: "Deployment history", "Manage Access", "Settings", and "...". The main area is divided into three horizontal stages: "Development", "Test", and "Production".

- Development:** A box containing the text "Design, review, and revise your content in a development workspace. When it's ready to test and preview, deploy the content to the test stage." Below this are workspace counts: Dataflows (0), Datacharts (0), Datasets (1), and Reports (1). Buttons include "Show more", "Publish app", and a green "Deploy" button.
- Test:** A box containing the text "Test and verify your content in a preproduction workspace. When it's ready to distribute, deploy the content to the production stage." Below this are workspace counts: Dataflows (0), Datacharts (0), Datasets (1), and Reports (1). Buttons include "Deploy to this stage or Assign a workspace" (with a dropdown menu "Select" and a link "Why can't I see all my workspaces?"), and "Assign a workspace".
- Production:** A box containing the text "Your content has been tested and is ready to distribute to your consumers as an app or by access to the production workspace." Below this are workspace counts: Dataflows (0), Datacharts (0), Datasets (1), and Reports (1). Buttons include "Deploy to this stage or Assign a workspace" (with a dropdown menu "Select" and a link "Why can't I see all my workspaces?"), and "Assign a workspace".

You can read more about creating deployment pipelines in [Deployment pipelines overview](#).

Step 5: Deploy content to other stages

Now, deploy the content to the other stages of the pipeline.

1. From the development stage of the deployment content view, select **Deploy**.

The screenshot shows the Microsoft Power BI Deployment Pipeline content view for the workspace "FoodSalesWS". The workspace summary shows:

- Dataflows: 0
- Datacharts: 0
- Datasets: 1
- Reports: 1

Below the summary are buttons for "Show more", "Publish app", and a prominent green "Deploy" button, which is highlighted with a red border. The "Deploy" button has a tooltip "Deploy to this stage or Assign a workspace".

2. Confirm that you want to deploy the content to the test stage.

Deploy from Development to Test

X

You are about to deploy from Development to Test. The items that are going to be deployed are:

New (3)

- FoodSales
- FoodSales
- FoodSales.pbix

Add a note ▾

Continue deployment in case 1 or more items fail [Learn more](#)

Deploy

Cancel

The green check icon indicates that the contents of the two stages are identical, since you deployed the entire content of the pipeline.

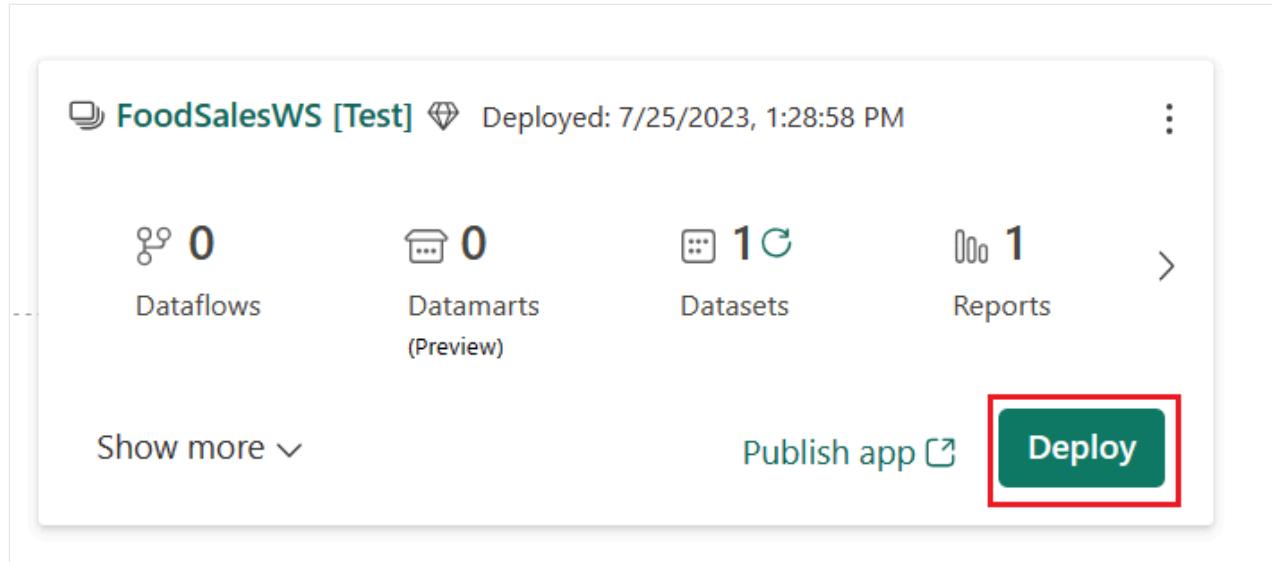
The screenshot shows the Power BI workspace interface with two stages: Development and Test.

Development Stage: Contains a summary of content items: 0 Dataflows, 0 Datacharts (Preview), 1 Dataset (highlighted with a red box), and 1 Report. Buttons include "Show more", "Publish app", and a green "Deploy" button.

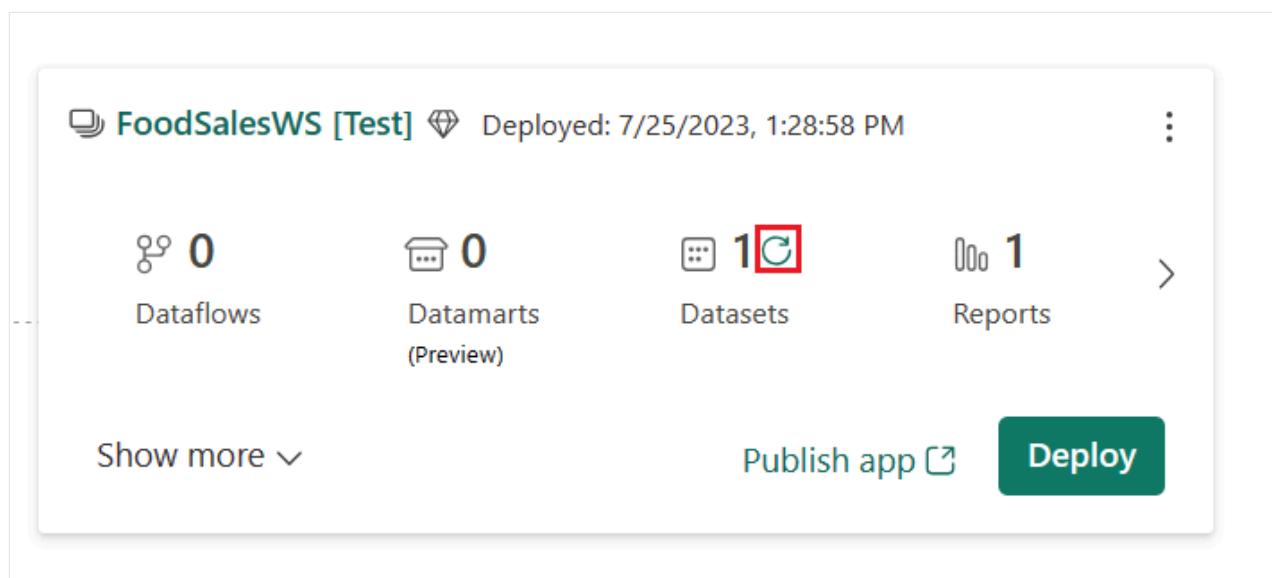
Test Stage: Contains a summary of content items: 0 Dataflows, 0 Datacharts (Preview), 1 Dataset (highlighted with a red box), and 1 Report. A green checkmark icon is displayed above the dataset count, indicating identical content. Buttons include "Show more", "Publish app", and a green "Deploy" button.

A central dashed arrow points from the Development stage to the Test stage, indicating the direction of deployment.

1. Deploy the content from the test stage to the production stage.



2. To refresh the semantic model in any stage, select the refresh button next to the semantic models icon in the summary card of each stage.



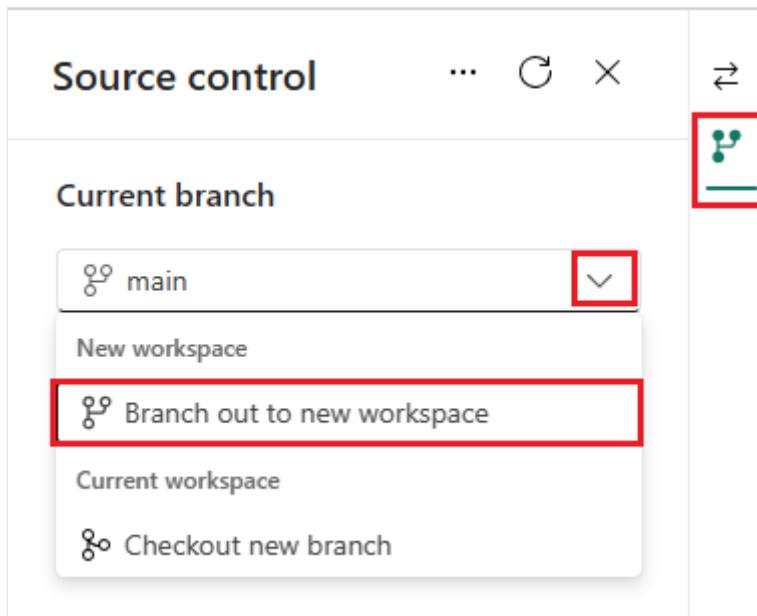
The entire team shares this deployment pipeline. Each team member can edit the semantic model and report in the development stage. When the team is ready to test the changes, they deploy the content to the test stage. When the team is ready to release the changes to production, they deploy the content to the production stage.

For more information on deploying content, see [Deploy content](#).

Step 6: Create an isolated workspace

In order to avoid editing the shared workspace and interfering with other team members' changes, each team member should create their own isolated workspace to work in until they're ready to share their changes with the team.

1. From the *branch* tab of the **Source control** menu, select the down arrow next to the current branch name, and select **Branch out to new workspace**.



2. Specify the following details about the branch and workspace. The new branch is automatically created based on the branch connected to the current workspace.

- Branch name (for this tutorial, name it *MyFoodEdits*)
- Workspace name (for this tutorial, name it *My_FoodSales*)

A screenshot of a dialog box titled "Branch out to a new workspace". The text inside says: "Create a copy of your current branch, connect it to a new workspace, and make changes without affecting your current workspace." The dialog contains several input fields and buttons.

- Branch name ***: The input field contains "MyFoodEdits".
- Based on ⓘ**: The input field contains "main".
- Workspace name ***: The input field contains "My_FoodSales".
- Buttons at the bottom:** A green button labeled "Branch out" is highlighted with a red box. To its right is a white button labeled "Cancel".

3. Select **Branch out**.

4. Select **Connect and sync**.

Fabric creates the new workspace and syncs it to the new branch. You're automatically taken to the new workspace, but the sync might take a few minutes.

The new workspace now contains the content of the Git repo folder. Notice it doesn't contain the *.pbix* file. Since *.pbix* files are unsupported, this file wasn't copied to the Git repo when we synced.

Use this workspace to make changes to the semantic model and report until you're ready to share them with your team.

Step 7: Edit the workspace

Once the branched out workspace is synced, you can make changes to the workspace by creating, deleting, or editing an item. In this tutorial, we change the format of a semantic model column. You can edit the workspace in [Power BI Desktop](#) or [data model](#). In this tutorial, we edit the workspace from the data model.

1. From the semantic model workspace, select the semantic model ellipsis (three dots) > [Open data model](#).

Name	Git status	Type	Owner
FoodSales	Synced	Report	My_FoodSales
FoodSales	Synced	Semantic model	My_FoodSales

A context menu is open for the second row, showing options: Analyze in Excel, Create report, Auto-create report, Create paginated report, Delete, Get quick insights, Security, Rename, Open data model (which is highlighted with a red box), Settings, Download this file, Manage permissions, and View lineage.

① Note

If **Open data model** is disabled, go to **Workspace settings > Power BI > General** and enable **Data model settings**.

Workspace settings

Search

About

Premium

Azure connections

System storage

Git integration

Other

Organization apps

Secure update

Allow contributors to update the app for this workspace

Allow contributors to update the app

Template apps

Template apps are developed for sharing outside your organization. A template app workspace will be created for developing and releasing the app. [Learn more about template apps](#)

Develop template apps

Data model settings

Data model settings

Power BI

General

Embed codes

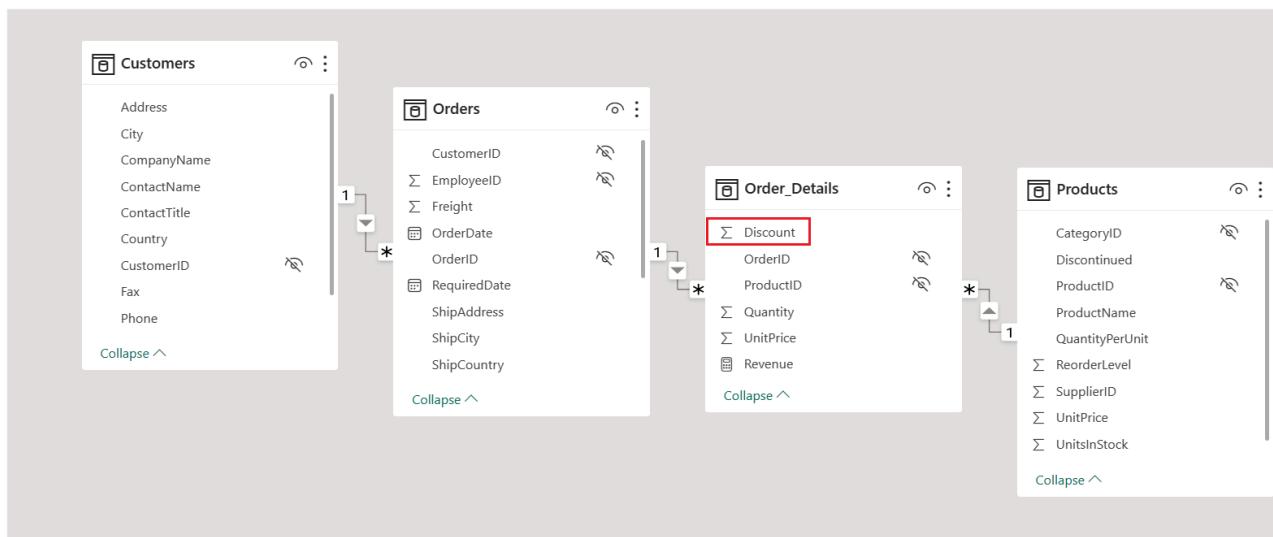
Allow workspace members to edit data models in the service. Edits are permanent and automatically saved in this feature preview, and version history isn't saved. This setting doesn't apply to DirectLake datasets or editing a dataset through an API or XMLA endpoint. [Learn more](#)

Users can edit data models in the Power BI service (preview)

Data

Engineering/Science

2. From the Order_Details table, select Discount.



3. From the Properties pane, change the Format from General to Percentage.

Properties »

General

Name
Discount

Description
Enter a description

Display folder
Enter the display folder

Is hidden
No

Formatting

Data type
Decimal number

Format
Percentage

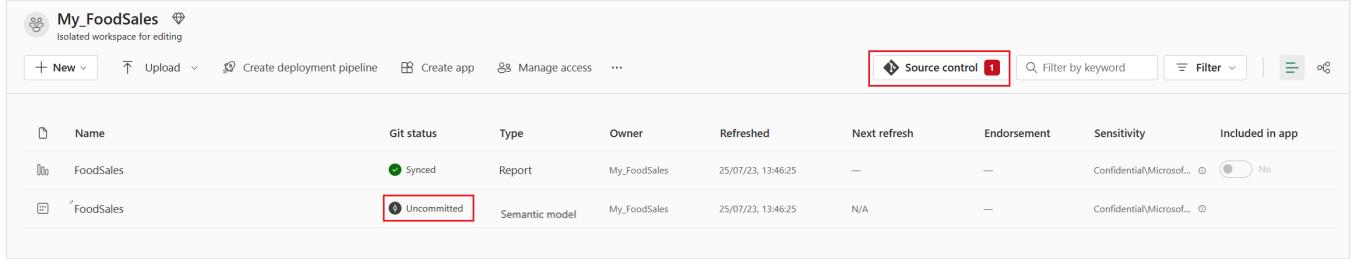
Percentage format
Yes

Thousands separator

Step 8: Commit changes

To commit this change from the workspace into the Git branch, go back to the workspace home page.

The source control icon now shows 1 because one item in the workspace was changed but not committed to the Git repo. The *FoodSales* semantic model shows a status of *Uncommitted*.



A screenshot of the Microsoft Power BI workspace interface. The top navigation bar includes 'New', 'Upload', 'Create deployment pipeline', 'Create app', 'Manage access', and a 'Source control' button with a red badge showing '1'. Below the navigation is a search bar 'Filter by keyword' and a 'Filter' dropdown. The main content area displays a table with two rows:

Name	Git status	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
FoodSales	Synced	Report	My_FoodSales	25/07/23, 13:46:25	—	—	Confidential\Microsoft...	<input checked="" type="checkbox"/> No
FoodSales	Uncommitted	Semantic model	My_FoodSales	25/07/23, 13:46:25	N/A	—	Confidential\Microsoft...	<input type="checkbox"/>

1. Select the source control icon to view the changed items in the Git repo. The semantic model shows a status of *Modified*.
2. Select the item to commit and add an optional message.
3. Select **Commit**.

Source control

C ... X

MyFoodEdits

⚠ Sensitivity labels of items are not reflected by metadata exported to Git.
Learn more [↗](#)

Changes 1 • Updates

Commit message

Add details before committing or we'll add for you by default.

Item

Status

FoodSales



The Git status of the semantic model changes to *Synced* and the workspace and Git repo are in sync.

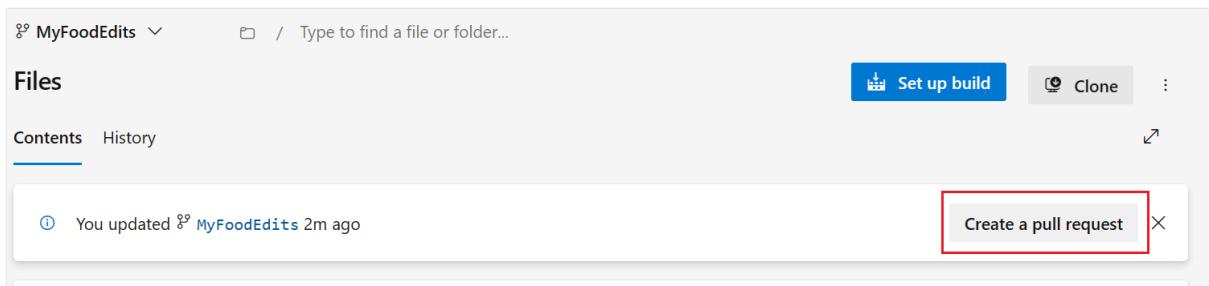
Step 9: Create PR and merge

In the Git repo, [create a pull request](#) to merge the *MyFoodEdits* branch with the *main* branch.

This step can be done manually or automated:

Manual merge PR

1. Select **Create a pull request**.



2. Provide a title, description, and any other information you want for the pull request.
Then select **Create**.

New pull request

MyFoodEdits into main

Overview Files 15 Commits 2

Title: My foods dataset test

Description: Testing changes in My Foods dataset
42/4000

Markdown supported Drag & drop, paste, or select files to insert.

Link work items.

Testing changes in My Foods Income dataset

Reviewers: Add required reviewers

Search users and groups to add as reviewers

Work items to link

Search work items by ID or title

Tags

Create

This screenshot shows the 'New pull request' interface. It includes fields for the title ('My foods dataset test'), description ('Testing changes in My Foods dataset'), and body text ('Testing changes in My Foods Income dataset'). There are sections for reviewers ('Add required reviewers') and work items ('Search work items by ID or title'). The 'Create' button at the bottom right is highlighted with a red box.

3. Merge the pull request.

Complete pull request

X

Merge type

Merge (no fast forward)



Post-completion options

Complete associated work items after merging

Delete MyFoods after merging

Customize merge commit message

Cancel

Complete merge

Once the changes have been merged to the main branch, you can safely delete the workspace, if you want. It's not deleted automatically.

Step 10: Update shared workspace

Go back to the shared workspace connected to the dev stage of the deployment pipeline (the one we created in [Step 1](#)) and refresh the page.

The source control icon now shows 1 because one item in the Git repo was changed and is different from the items in the FoodSales workspace. The FoodSales semantic model shows a status of *Update required*.

You have unsynced commits from Git. We recommend that you update from Git before you continue working.

Name	Git status	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
FoodSales	Synced	Report	FoodSalesWS	25/07/23, 12:48:56	—	—	Non-Business	No
FoodSales	Update Required	Semantic model	FoodSalesWS	25/07/23, 12:48:56	N/A	—	Non-Business	No
FoodSales.pbix	Unsupported	Dashboard	FoodSalesWS	—	—	—	—	No

You can update the workspace manually or automated:

Manual update workspace

1. Select the source control icon to view the changed items in the Git repo. The semantic model shows a status of Modified.
2. Select **Update all**.

The screenshot shows the 'Source control' interface. At the top left is a 'Source control' button with a red notification badge containing the number '1'. To its right is a search bar labeled 'Filter by keyword' and a 'Filter' dropdown. On the far right are three icons: a grid, a magnifying glass, and a refresh symbol. The main area is titled 'Source control' and shows a dropdown menu set to 'main'. Below this, there are two tabs: 'Changes' and 'Updates 1 •', with 'Updates' being the active tab. A table lists items with columns for 'Item' and 'Status'. One item, 'FoodSales', has a status icon with a red border. At the bottom is a green button labeled 'Update all' with a red border.

The Git status of the semantic model changes to *Synced* and the workspace is synced with the *main* Git branch.

Step 11: Compare stages in deployment pipeline

1. Select **View deployment pipeline** to compare the content in the development stage with the content in the test stage.

The screenshot shows the Azure Data Studio interface with the following details:

- Top Bar:** Includes icons for New, Upload, View deployment pipeline (which is highlighted with a red box), Create app, Manage access, and more.
- Table:** A list of items under the heading "FoodSalesWS". It includes columns for Name, Git status, Type, Owner, and Refreshed date.

	Name	Git status	Type	Owner	Refreshed
FoodSales	Synced	Report	FoodSalesWS	25/07/23, 12:48:56	
FoodSales	Synced	Dataset	FoodSalesWS	25/07/23, 12:48:56	
FoodSales.pbix	Unsupported	Dashboard	FoodSalesWS	—	

Notice the orange between the stages indicating that changes were made to the content in one of the stages since the last deployment.

The screenshot shows two stages in a deployment pipeline:

- Test Stage:** Contains 0 Dataflows, 0 Datamarts (Preview), 1 Dataset, and 1 Report. It has a "Deploy" button.
- Development Stage:** Contains 0 Dataflows, 0 Datamarts (Preview), 1 Dataset, and 1 Report. It also has a "Deploy" button.

A comparison window is open between the two stages, with the "Review changes" button highlighted with a red box.

2. Select the down arrow > Review Changes to view the changes. The Change Review screen shows the difference between the semantic models in the two stages.

The screenshot shows a detailed comparison of semantic models between the Test and Development stages. The interface displays code snippets for both stages, with differences highlighted in red.

Item list [1]	FoodSalesWS / FoodSales	-1 +1
FoodSales	To be modified [Test]	To be deployed [Development]
	<pre> 1214], 1215 "showAsVariationsOnly": true 1216 }, 1217 { 1218 "name": "Order_Details", 1219 "annotations": [1220 { 1221 "name": "PBI_ResultType", 1222 "value": "Table" 1223 } 1224], 1225 "columns": [1226 { 1227 "name": "Discount", 1228 "annotations": [1229 { 1230 "name": "SummarizationSetBy", 1231 "value": "Automatic" 1232 } 1233], 1234 "dataType": "double", 1235 "formatString": "0.00%;-0.00%;0.00%", 1236 "lineageTag": "e6ca9cf4-89aa-429f-9cad-416961fb4e43", 1237 "sourceColumn": "Discount", 1238 "summarizeBy": "sum" 1239 }, 1240 { 1241 "name": "OrderID", 1242 "annotations": [1243 { 1244 "name": "SummarizationSetBy", 1245 "value": "Automatic" 1246 } 1247], 1248 "dataType": "double", 1249 "formatString": "0", 1250 "lineageTag": "e6ca9cf4-89aa-429f-9cad-416961fb4e43", 1251 "sourceColumn": "OrderID", 1252 "summarizeBy": "sum" 1253 } 1254], 1255 "keyColumn": "OrderID" 1256 } 1257 }</pre>	<pre> 1214], 1215 "showAsVariationsOnly": true 1216 }, 1217 { 1218 "name": "Order_Details", 1219 "annotations": [1220 { 1221 "name": "PBI_ResultType", 1222 "value": "Table" 1223 } 1224], 1225 "columns": [1226 { 1227 "name": "Discount", 1228 "annotations": [1229 { 1230 "name": "SummarizationSetBy", 1231 "value": "Automatic" 1232 } 1233], 1234 "dataType": "double", 1235 "formatString": "0", 1236 "lineageTag": "e6ca9cf4-89aa-429f-9cad-416961fb4e43", 1237 "sourceColumn": "Discount", 1238 "summarizeBy": "sum" 1239 }, 1240 { 1241 "name": "OrderID", 1242 "annotations": [1243 { 1244 "name": "SummarizationSetBy", 1245 "value": "Automatic" 1246 } 1247], 1248 "dataType": "double", 1249 "formatString": "0", 1250 "lineageTag": "e6ca9cf4-89aa-429f-9cad-416961fb4e43", 1251 "sourceColumn": "OrderID", 1252 "summarizeBy": "sum" 1253 } 1254], 1255 "keyColumn": "OrderID" 1256 } 1257 }</pre>

3. Review the changes and close the window.

For more information about comparing stages in a deployment pipeline, see [Compare stages in a deployment pipeline](#).

Step 12: Deploy to test stage

When you're satisfied with the changes, deploy the changes to the test and/or production stages using the same process you used in [Step 5](#).

Summary

In this tutorial, you learned how to use deployment pipelines along with Git integration to manage the lifecycle of an app, report, or other content in a workspace.

In particular, you learned how to:

- Set up your workspaces and add content for managing their lifecycle in Fabric.
- Apply Git best practices to work alone and collaborate with teammates on changes.
- Combine Git and deployment pipelines for an efficient end to end release process.

Related content

- [Power BI Desktop projects and Azure DevOps build pipelines](#)
- [Manage Git branches](#)

Last updated on 12/15/2025

Choose the best Fabric CI/CD workflow option for you

The goal of this article is to present Fabric developers with different options for building CI/CD processes in Fabric, based on common customer scenarios. This article focuses more on the *continuous deployment* (CD) of the CI/CD process. For a discussion on the *continuous integration* (CI) part, see [Manage Git branches](#).

While this article outlines several distinct options, many organizations take a hybrid approach.

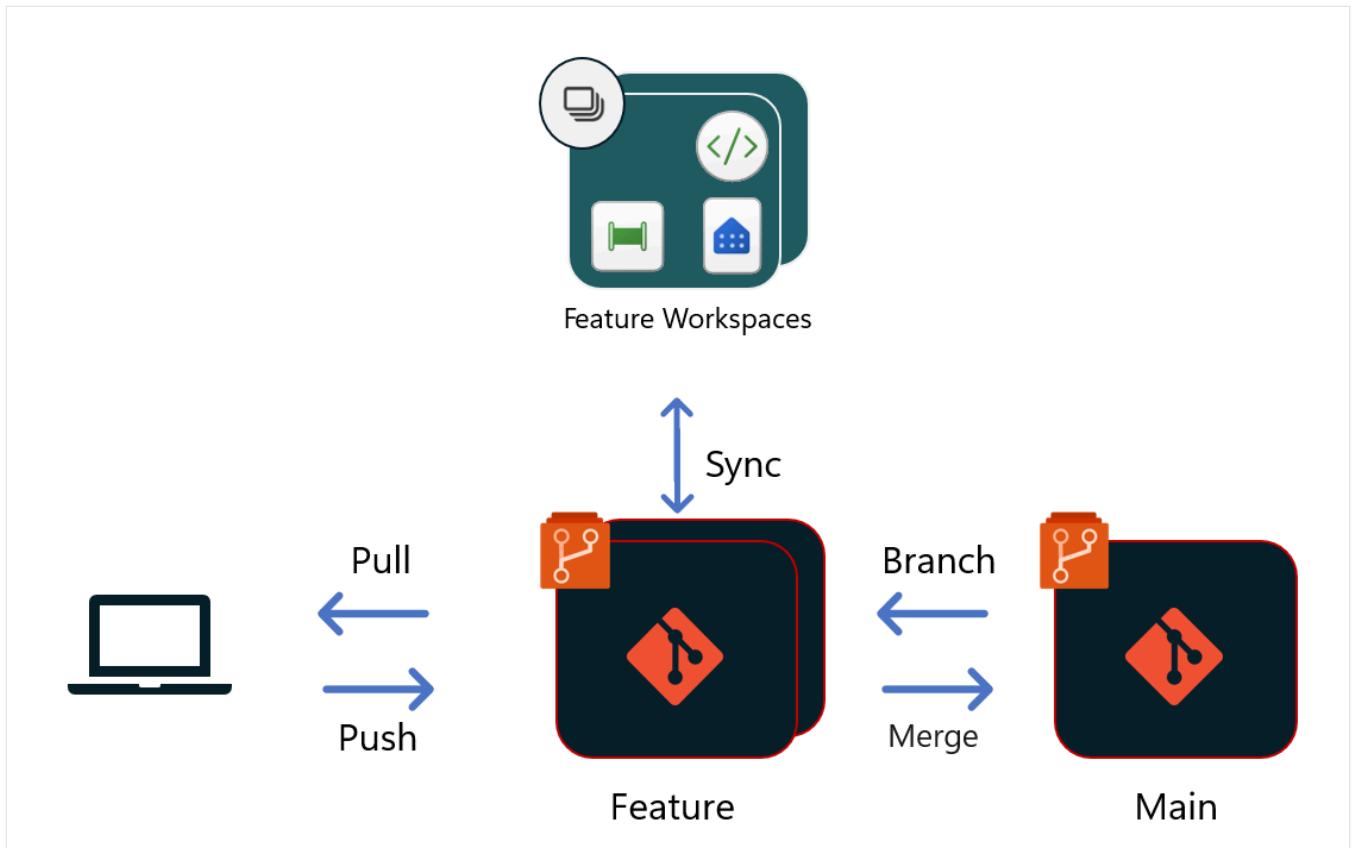
Prerequisites

To access the deployment pipelines feature, you must meet the following conditions:

- You have a [Microsoft Fabric subscription](#)
- You're an admin of a Fabric [workspace](#)

Development process

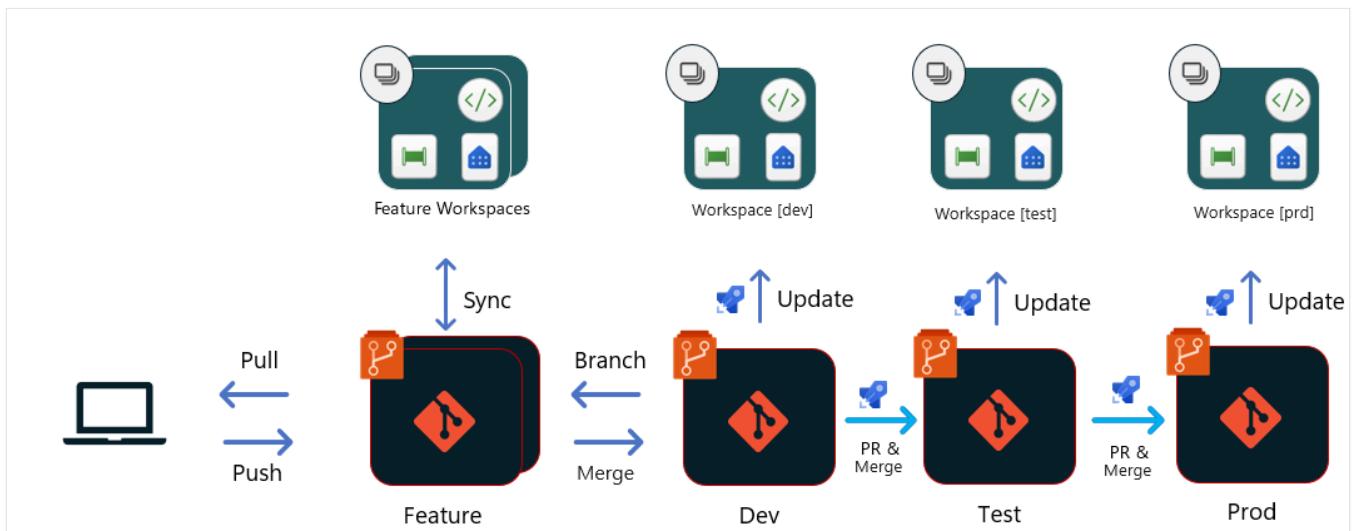
The development process is the same in all deployment scenarios, and is independent of how to release new updates into production. When developers work with source control, they need to work in an isolated environment. In Fabric, that environment can either be an IDE on your local machine (such as Power BI Desktop, or VS Code), or a different workspace in Fabric. You can find information about the different considerations for the development process in [Manage Git branches](#)



Release process

The release process starts once new updates are complete and the pull request (PR) merged into the team's shared branch (such as *Main*, *Dev* etc.). From this point, there are different options to build a release process in Fabric.

Option 1 - Git- based deployments



With this option, all deployments originate from the Git repository. Each stage in the release pipeline has a dedicated primary branch (in the diagram, these stages are *Dev*, *Test*, and *Prod*), which feeds the appropriate workspace in Fabric.

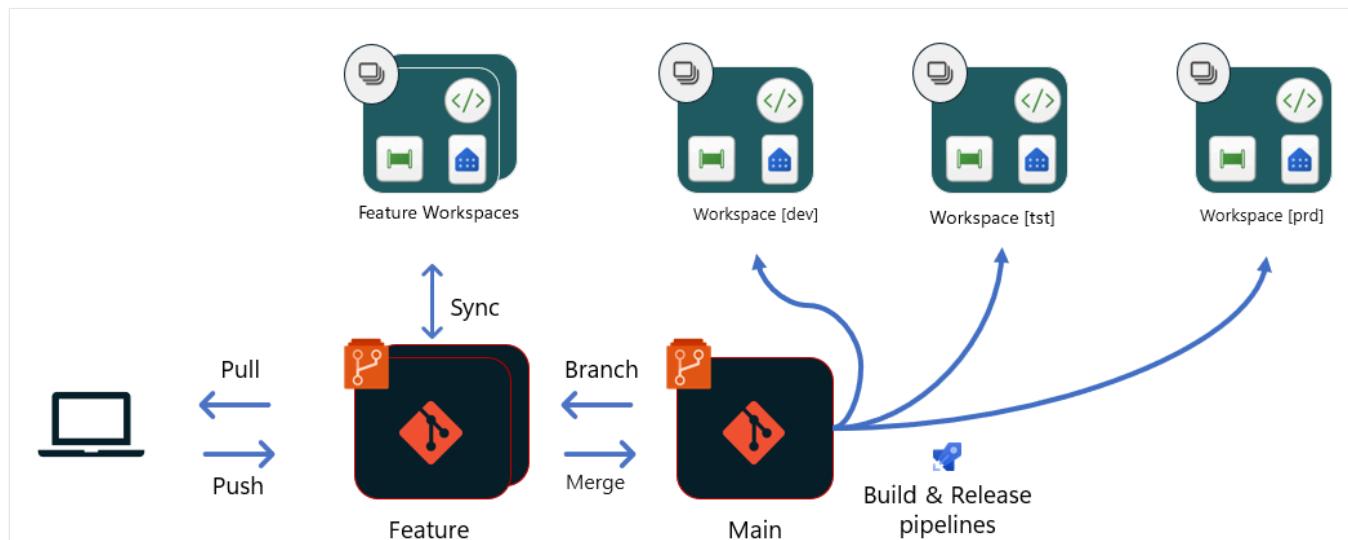
Once a PR to the *Dev* branch is approved and merged:

1. A release pipeline is triggered to update the content of the *Dev* workspace. This process also can include a *Build* pipeline to run unit tests, but the actual upload of files is done directly from the repo into the workspace, using [Fabric Git APIs](#). You might need to call other Fabric APIs for post-deployment operations that set specific configurations for this workspace, or ingest data.
2. A PR is then created to the *Test* branch. In most cases, the PR is created using a release branch that can cherry pick the content to move into the next stage. The PR should include the same review and approval processes as any other in your team or organization.
3. Another *Build* and *release* pipeline is triggered to update the *Test* workspace, using a process similar to the one described in the first step.
4. A PR is created to *Prod* branch, using a process similar to the one described in step #2.
5. Another *Build* and *release* pipeline is triggered to update the *Prod* workspace, using a process similar to the one described in the first step.

When should you consider using option #1?

- When you want to use your Git repo as the single source of truth, and the origin of all deployments.
- When your team follows *Gitflow* as the branching strategy, including multiple primary branches.
- The upload from the repo goes directly into the workspace, as we don't need *build environments* to alter the files before deployments. You can change this by calling APIs or running items in the workspace after deployment.

Option 2 - Git- based deployments using Build environments



With this option, all deployments originate from the same branch of the Git repository (*Main*). Each stage in the release pipeline has a dedicated *build* and *Release* pipeline. These pipelines might use a *Build environment* to run unit tests and scripts that change some of the definitions in the items before they're uploaded to the workspace. For example, you might want to change the data source connection, the connections between items in the workspace, or the values of parameters to adjust configuration for the right stage.

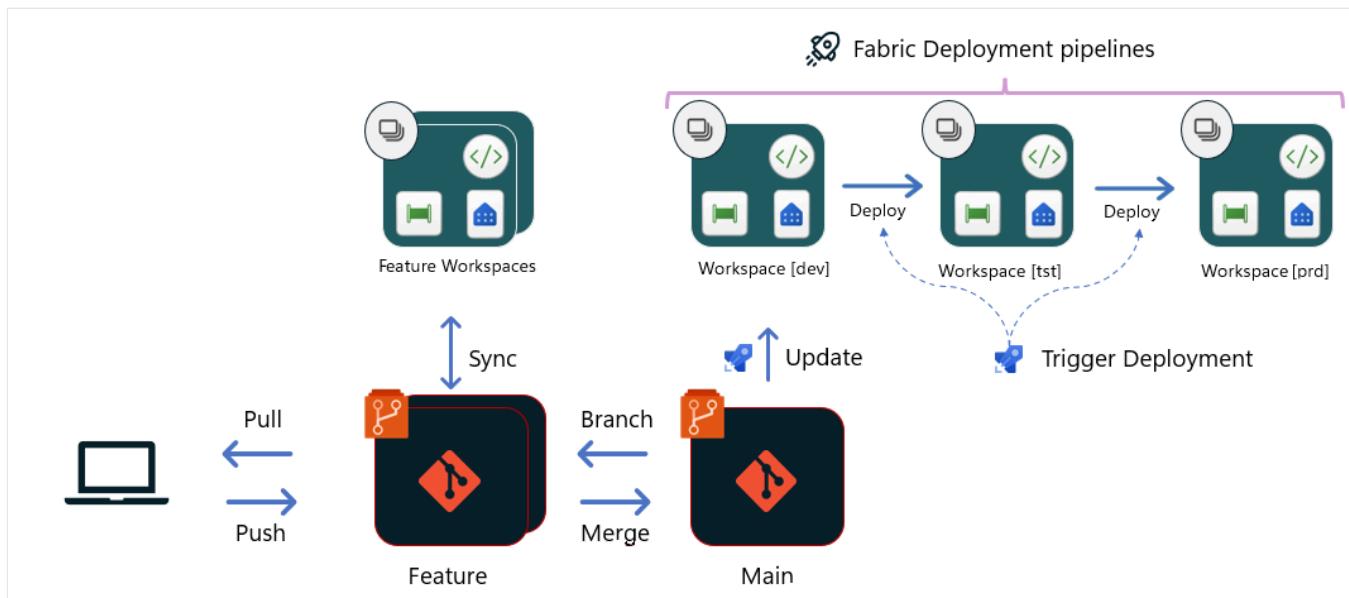
Once a PR to the *dev* branch is approved and merged:

1. A *build* pipeline is triggered to spin up a new *Build environment* and run unit tests for the *dev* stage. Then, a *release* pipeline is triggered to upload the content to a *Build environment*, run scripts to change some of the configuration, adjust the configuration to *dev* stage, and use Fabric's [Update item definition](#) APIs to upload the files into the Workspace.
2. When this process is complete, including ingesting data and approval from release managers, the next *build* and *release* pipelines for *test* stage can be created. These stages are created in a process similar to that described in the first step. For *test* stage, other automated or manual tests might be required after the deployment, to validate the changes are ready to be released to *Prod* stage.
3. When all automated and manual tests are complete, the release manager can approve and kick off the *build* and *release* pipelines to *Prod* stage. As the *Prod* stage usually has different configurations than *test/Dev* stages, it's important to also test out the changes after the deployment. Also, the deployment should trigger any more ingestion of data, based on the change, to minimize potential non availability to consumers.

When should you consider using option #2?

- When you want to use Git as your single source of truth, and the origin of all deployments.
- When your team follows *Trunk-based* workflow as its branching strategy.
- You need a build environment (with a custom script) to alter workspace-specific attributes, such as *connectionId* and *lakehouseld*, before deployment.
- You need a release pipeline (custom script) to retrieve item content from git and call the corresponding [Fabric Item API](#) for creating, updating, or deleting modified Fabric Items.

Option 3 - Deploy using Fabric deployment pipelines



With this option, Git is connected only until the *dev* stage. From the *dev* stage, deployments happen directly between the workspaces of *Dev/Test/Prod*, using Fabric deployment pipelines. While the tool itself is internal to Fabric, developers can use the [deployment pipelines APIs](#) to orchestrate the deployment as part of their Azure release pipeline, or a GitHub workflow. These APIs enable the team to build a similar *build* and *release* process as in other options, by using automated tests (that can be done in the workspace itself, or before *dev* stage), approvals etc.

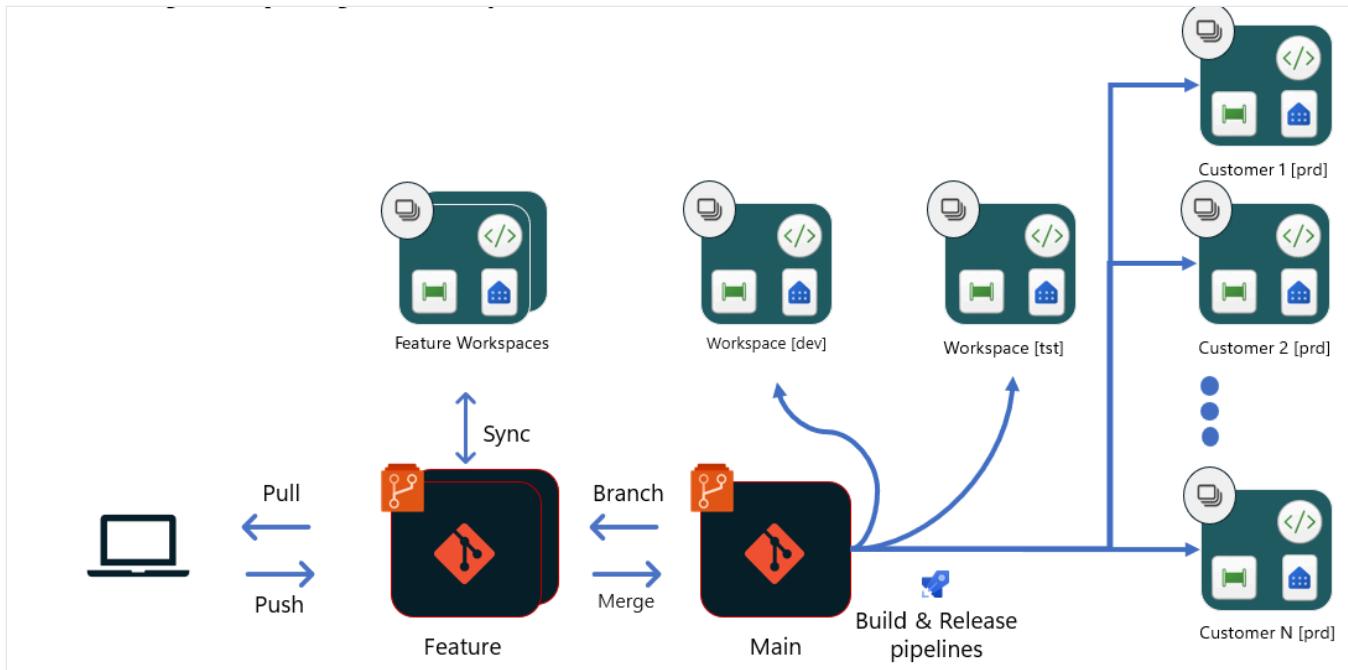
Once the PR to the *main* branch is approved and merged:

1. A *build* pipeline is triggered that uploads the changes to the *dev* stage using [Fabric Git APIs](#). If necessary, the pipeline can trigger other APIs to start post-deployment operations/tests in the *dev* stage.
2. After the *dev* deployment is completed, a release pipeline kicks in to deploy the changes from *dev* stage to *test* stage. Automated and manual tests should take place after the deployment, to ensure that the changes are well-tested before reaching production.
3. After tests are completed and the release manager approves the deployment to *Prod* stage, the release to *Prod* kicks in and completes the deployment.

When should you consider using option #3?

- When you're using source control only for development purposes, and prefer to deploy changes directly between stages of the release pipeline.
- When deployment rules, autobinding and other available APIs are sufficient to manage the configurations between the stages of the release pipeline.
- When you want to use other functionalities of Fabric deployment pipelines, such as viewing changes in Fabric, deployment history etc.
- Consider also that deployments in Fabric deployment pipelines have a linear structure, and require other permissions to create and manage the pipeline.

Option 4 - CI/CD for ISVs in Fabric (managing multiple customers/solutions)



This option is different from the others. It's most relevant for Independent Software Vendors (ISV) who build SaaS applications for their customers on top of Fabric. ISVs usually have a separate workspace for each customer and can have as many as several hundred or thousands of workspaces. When the structure of the analytics provided to each customer is similar and out-of-the-box, we recommend having a centralized development and testing process that splits off to each customer only in the *Prod* stage.

This option is based on [option #2](#). Once the PR to *main* is approved and merged:

1. A *build* pipeline is triggered to spin up a new *Build environment* and run unit tests for *dev* stage. When tests are complete, a *release* pipeline is triggered. This pipeline can upload the content to a *Build environment*, run scripts to change some of the configuration, adjust the configuration to *dev* stage, and then use Fabric's [Update item definition](#) APIs to upload the files into the Workspace.
2. After this process is complete, including ingesting data and approval from release managers, the next *build* and *release* pipelines for *test* stage can kick off. This process is similar to that described in the first step. For *test* stage, other automated or manual tests might be required after the deployment, to validate the changes are ready to be released to *Prod* stage in high-quality.
3. Once all tests pass and the approval process is complete, the deployment to *Prod* customers can start. Each customer has its own release with its own parameters, so that its specific configuration and data connection can take place in the relevant customer's workspace. The configuration change can happen through scripts in a *build* environment,

or using APIs post deployment. All releases can happen in parallel as they aren't related nor dependent of each other.

When should you consider using option #4?

- You're an ISV building applications on top of Fabric.
- You're using different workspaces for each customer to manage the multi-tenancy of your application
- For more separation, or for specific tests for different customers, you might want to have multi-tenancy in earlier stages of *dev* or *test*. In that case, consider that with multi-tenancy the number of workspaces required grows significantly.

Summary

This article summarizes the main CI/CD options for a team who wants to build an automated CI/CD process in Fabric. While we outline four options, the real-life constraints and solution architecture might lend themselves to hybrid options, or completely different ones. You can use this article to guide you through different options and how to build them, but you're not forced to choose only one of the options.

Some scenarios or specific items might have [limitations in place](#) that can keep you from adopting any of these scenarios.

The same goes for tooling. While we mention different tools here, you might choose other tools that can provide same level of functionality. Consider that Fabric has better integration with some tools, so choosing others result in more limitations that need different workarounds.

Related content

- [Manage Git branches](#)
- [Automate Git integration by using APIs and Azure DevOps](#)
- [Automate deployment pipeline by using Fabric APIs](#)
- [Best practices for lifecycle management in Fabric](#)

What is Microsoft Fabric Git integration?

This article explains to developers how to integrate Git version control with the Microsoft Fabric Application lifecycle management (ALM) tool.

 Note

Some of the items for Git integration are in preview. For more information, see the list of [supported items](#).

Git integration in Microsoft Fabric enables developers to integrate their development processes, tools, and best practices straight into the Fabric platform. It allows developers who are developing in Fabric to:

- Backup and version their work
- Revert to previous stages as needed
- Collaborate with others or work alone using Git branches
- Apply the capabilities of familiar source control tools to manage Fabric items

The integration with source control is on a workspace level. Developers can version items they develop within a workspace in a single process, with full visibility to all their items. The workspace structure, including [subfolders](#), is preserved in the Git repository.

See the list of [supported items](#).

- Read up on basic [Git](#) and [version control](#) concepts.
- Read more about the [Git integration process](#).
- Read about the best way to manage your [Git branches](#).

Network security for Git integration

Workspace-level security in Microsoft Fabric provides granular control over data access and network connectivity by allowing administrators to configure both inbound and outbound protections for individual workspaces. These controls ensure that sensitive data remains within trusted network boundaries, and they integrate with CI/CD tools like Git integration. For more information, see [Network security for continuous integration/continuous deployment](#)

Privacy information

Before you enable Git integration, make sure you review the following privacy statements:

- Microsoft privacy statement [↗](#)
- Azure DevOps Services Data protection overview
- GitHub Data protection agreement [↗](#)

Supported Git providers

The following Git providers are supported:

- [Azure DevOps](#) (cloud-based only)
- [GitHub](#) [↗](#) (cloud-based only)
- [GitHub Enterprise](#) [↗](#) (cloud-based only)

Supported items

The following items currently support Git integration:

- Data Engineering items:
 - [Environment](#)
 - [GraphQL](#)
 - [Lakehouse](#) (*preview*)
 - [Notebooks](#)
 - [Spark Job Definitions](#)
 - [User Data Functions](#)
- Data Science items:
 - [Machine learning experiments](#) (*preview*)
 - [Machine learning models](#) (*preview*)
 - [Data Agents](#) (*preview*)
- Data Factory items:
 - [Copy Job](#)
 - [Dataflow gen2](#)
 - [Pipeline](#)
 - [Mirrored database](#)
 - [Mount ADF](#)
 - [Mirrored snowflake](#) (*preview*)
- Real-time Intelligence items:
 - [Activator](#) (*preview*)
 - [Eventhouse](#)
 - [EventStream](#)
 - [KQL database](#)

- [KQL Queryset](#)
 - [Real-time Dashboard](#)
 - [Event Schema Set \(preview\)](#)
 - [Maps \(preview\)](#)
 - [Anomaly detection \(preview\)](#)
- Data Warehouse items:
 - [Warehouse \(preview\)](#)
 - Mirrored Azure Databricks Catalog
- Power BI items:
 - Metrics Set (preview)
 - [Org app \(preview\)](#)
 - [Paginated report \(preview\)](#)
 - Report (except reports connected to semantic models hosted in [Azure Analysis Services](#), [SQL Server Analysis Services](#), or reports exported by Power BI Desktop that depend on semantic models hosted in [MyWorkspace](#)) (preview)
 - [Semantic model](#) (except push datasets, live connections to Analysis Services, model v1) (preview)
- Database items:
 - [SQL database](#)
 - [Cosmos database \(preview\)](#)
- Graph:
 - [Graph in Microsoft Fabric \(preview\)](#)
- Industry solutions:
 - [Healthcare \(preview\)](#)
 - [HealthCare Cohort \(preview\)](#)

If the workspace or Git directory has unsupported items, it can still be connected, but the unsupported items are ignored. They aren't saved or synced, but they're not deleted either. They appear in the source control panel but you can't commit or update them.

Considerations and limitations

General Git integration limitations

- The [authentication method](#) in Fabric must be at least as strong as the authentication method for Git. For example, if Git requires multifactor authentication, Fabric needs to require multifactor authentication as well.

- Power BI Datasets connected to Analysis Services aren't supported at this time.
- If you use a workspace identity in one artifact and commit it to Git, it can be updated (back to a fabric workspace) only in a workspace connected to the same identity. Be careful, as this also affects features like branch out.
- Submodules aren't supported.
- Sovereign clouds aren't supported.
- If your workspace contains hundreds of items, consider splitting it into smaller sets of artifacts. Each set should be placed in a separate workspace and linked to a different Git branch, or connected to a single branch organized into different folders.

Azure DevOps limitations

- Azure DevOps isn't supported if [Enable IP Conditional Access policy validation](#) is enabled.
- If the workspace and Git repo are in two different geographical regions, the tenant admin must enable [cross-geo exports](#).
- If your organization configured [conditional access](#), make sure the **Power BI Service** has the same [conditions set](#) for authentication to function as expected.
- The following commit size limit is applied:
 - 25 MB using the Azure DevOps connector with Service Principal.
 - 125 MB using the default single sign-on (SSO) Microsoft Entra ID account and Azure DevOps connector with User Principal.

GitHub Enterprise limitations

Some GitHub Enterprise versions and settings aren't supported. For example:

- GitHub Enterprise Cloud with data residency (ghe.com)
- GitHub Enterprise Server with a custom domain is not supported, even if the instance is publicly accessible
- GitHub Enterprise Server hosted on a private network
- IP allowlist

Azure DevOps to GitHub Enterprise migration consideration

If your team uses Fabric Git Integration and is evaluating a migration from Azure DevOps to GitHub Enterprise, it's recommended to run validation tests to ensure Git Integration functionality remains unaffected. Fabric Git Integration relies on the underlying Git provider APIs, which differ in capabilities and limitations between Azure DevOps and GitHub Enterprise, as described above.

Workspace limitations

- Only the workspace admin can manage the connections to the [Git Repo](#) such as connecting, disconnecting, or adding a branch.
Once connected, anyone with [permission](#) can work in the workspace.
- Workspaces with template apps installed can't be connected to Git.
- [MyWorkspace](#) can't connect to a Git provider.

Branch and folder limitations

- Maximum length of branch name is 244 characters.
- Maximum length of full path for file names is 250 characters. Longer names fail.
- Maximum file size is 25 MB.
- Folder structure is maintained up to 10 levels deep.
- Downloading a report/dataset as [.pbix](#) from the service after deploying them with Git integration is not recommended, as the results are unreliable. We recommend using PowerBI Desktop to download reports/datasets as [.pbix](#).
- If the item's display name has any of these characteristics, The Git folder is renamed to the logical ID (Guid) and type:
 - Has more than 256 characters
 - Ends with a [.](#) or a space
 - Contains any forbidden characters as described in [directory name limitations](#)
- When you connect a workspace that has folders to Git, you need to commit changes to the Git repo if that [folder structure](#) is different.

Directory name limitations

- The name of the directory that connects to the Git repository has the following naming restrictions:
 - The directory name can't begin or end with a space or tab.
 - The directory name can't contain any of the following characters: ["](#) [/](#) [:](#) [<](#) [>](#) [\](#) [*](#) [?](#) [|](#)
- The item folder (the folder that contains the item files) can't contain any of the following characters: ["](#) [:](#) [<](#) [>](#) [\](#) [*](#) [?](#) [|](#). If you rename the folder to something that includes one of these characters, Git can't connect or sync with the workspace and an error occurs.

Branching out limitations

- Branch out requires permissions listed in [permissions table](#).
- There must be an available capacity for this action.

- All [workspace](#) and [branch naming limitations](#) apply when branching out to a new workspace.
- Only [Git supported items](#) are available in the new workspace.
- The related branches list only shows branches and workspaces you have permission to view.
- [Git integration](#) must be enabled.
- When branching out, a new branch is created and the settings from the original branch aren't copied. Adjust any settings or definitions to ensure that the new meets your organization's policies.
- When branching out to an existing workspace:
 - The target workspace must support a Git connection.
 - The user must be an admin of the target workspace.
 - The target workspace must have capacity.
 - The workspace can't have template apps.
- Note that when you branch out to a workspace, any items that aren't saved to Git can get lost. We recommend that you [commit](#) any items you want to keep before branching out.

Sync and commit limitations

- You can only sync in one direction at a time. You can't commit and update at the same time.
- Sensitivity labels aren't supported and exporting items with sensitivity labels might be disabled. To commit items that have sensitivity labels without the sensitivity label, [ask your administrator](#) for help.
- Works with [limited items](#). Unsupported items in the folder are ignored.
- Duplicating names isn't allowed. Even if Power BI allows name duplication, the update, commit, or undo action fails.
- B2B isn't supported.
- [Conflict resolution](#) is partially done in Git.
- During the *Commit to Git* process, the Fabric service deletes files *inside the item folder* that aren't part of the item definition. Unrelated files not in an item folder aren't deleted.
- After you commit changes, you might notice some unexpected changes to the item that you didn't make. These changes are semantically insignificant and can happen for several reasons. For example:
 - Manually changing the item definition file. These changes are valid, but might be different than if done through the editors. For example, if you rename a semantic model column in Git and import this change to the workspace, the next time you commit changes to the semantic model, the *bim* file will register as changed and the modified column pushed to the back of the `columns` array. This is because the AS

engine that generates the *bim* files pushes renamed columns to the end of the array.

This change doesn't affect the way the item operates.

- Committing a file that uses *CRLF* line breaks. The service uses *LF* (line feed) line breaks.
If you had item files in the Git repo with *CRLF* line breaks, when you commit from the service these files are changed to *LF*. For example, if you open a report in desktop, save the project file (*.pbip*) and upload it to Git using *CRLF*.
- Refreshing a semantic model using the [Enhanced refresh API](#) causes a Git diff after each refresh.

Related content

- [Get started with Git integration](#)
- [Understand the Git integration process](#)

Last updated on 12/15/2025

Get started with Git integration

This article walks you through the following basic tasks in Microsoft Fabric's Git integration tool:

- [Connect to a Git repo](#)
- [Commit changes](#)
- [Update from Git](#)
- [Disconnect from Git](#)

We recommend reading the [overview of Git integration](#) before you begin.

Prerequisites

To integrate Git with your Microsoft Fabric workspace, you need to set up the following prerequisites for both Fabric and Git.

Fabric prerequisites

To access the Git integration feature, you need a [Fabric capacity](#). A Fabric capacity is required to use all supported Fabric items. If you don't have one yet, [sign up for a free trial](#). Customers that already have a [Power BI Premium capacity](#), can use that capacity, but keep in mind that [certain Power BI SKUs only support Power BI items](#).

In addition, the following [tenant switches](#) must be enabled from the Admin portal:

- [Users can create Fabric items](#)
- [Users can synchronize workspace items with their Git repositories](#)
- [Create workspaces](#) (only if you want to branch out to a new workspace.)
- [Users can synchronize workspace items with GitHub repositories](#): For GitHub users only

These switches can be enabled by the tenant admin, capacity admin, or workspace admin, depending on your [organization's settings](#).

Git prerequisites

Git integration is currently supported for Azure DevOps and GitHub. To use Git integration with your Fabric workspace, you need the following in either Azure DevOps or GitHub:

Azure DevOps

- An Active **Azure DevOps account** registered to same Fabric user (supported even if Azure DevOps organization reside in a different tenant than Fabric tenant). [Create a free account ↗](#).
- Access to an existing repository.

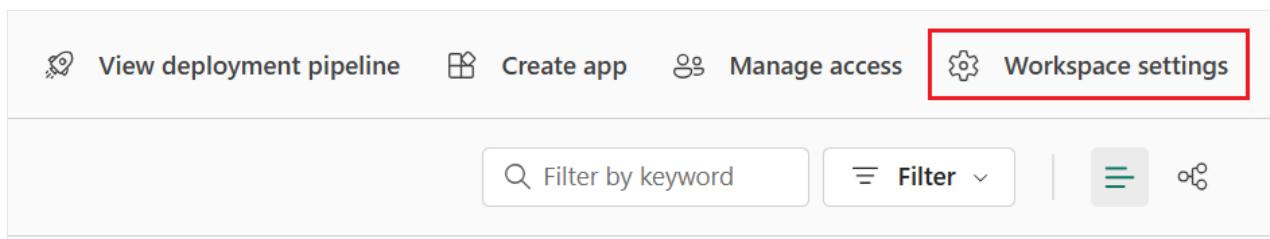
Connect a workspace to a Git repo

Connect to a Git repo

Only a workspace admin can connect a workspace to a repository, but once connected, anyone with [permission](#) can work in the workspace. If you're not an admin, ask your admin for help with connecting. To connect a workspace to an Azure or GitHub Repo, follow these steps:

1. Sign into Fabric and navigate to the workspace you want to connect with.

2. Go to **Workspace settings**



3. Select **Git integration**.

4. Select your Git provider. Currently, Azure DevOps and GitHub are supported.



If you select Azure DevOps, select **Connect** to automatically sign into the Azure Repos account registered to the Microsoft Entra user signed into Fabric.

If you have already signed in to Azure from Fabric using a different account, select your account from the list and select **Connect**.

If it's your first time signing in from Fabric, or you want to add a new account, select **Add account**.

If it's the first time connecting, you need to Authorize your user. Provide the following information:

- *Display name* - must be unique for each user
- *Azure DevOps URL* - URL of the Azure DevOps repository. URL must be in the format
`https://dev.azure.com/{organization}/{project}/_git/{repository}` or
`https://{{organization}}.visualstudio.com/{project}/_git/{repo}`.
- *Authentication* - You can authenticate either with *OAuth2* or a *Service Principal*. For more information see [Azure DevOps - Git Integration with service principal](#)

Add Azure DevOps account

Display name *

Azure DevOps URL * ⓘ

Authentication method * ⓘ

OAuth 2.0

Service principal

After you sign in, select **Connect** to allow Fabric to access your account

Connect to a workspace

If the workspace is already connected to Azure DevOps/GitHub, follow the instructions for [Connecting to a shared workspace](#).

Azure DevOps branch connect

1. From the dropdown menu, specify the following details about the branch you want to connect to:
 - [Organization](#)
 - [Project](#)
 - [Git repository](#).

- Branch (Select an existing branch using the drop-down menu, or select **+ New Branch** to create a new branch. You can only connect to one branch at a time.)
- Folder (Type in the name of an existing folder or enter a name to create a new folder. If you leave the folder name blank, content is created in the root folder. You can only connect to one folder at a time.)

Workspace settings

Search

Git integration

Connect to Git to manage your code and back up your work. [Learn more](#)

General

License info

Azure connections

System storage

Git integration

OneLake

Workspace identity

Network security

Encryption (preview)

Monitoring

Power BI

Delegated Settings

Data

Engineering/Science

Data Factory

Preview items Some item types are only available in preview when using Git. [Learn more](#)

Connect Git provider and account

Provider

Azure DevOps

Entra Account

Log out

Manage all accounts

Connect Git repository and branch

Organization *

Organization

Project *

Project

Git repository * ⓘ

Git repository

Branch * ⓘ

Branch

Git folder ⓘ

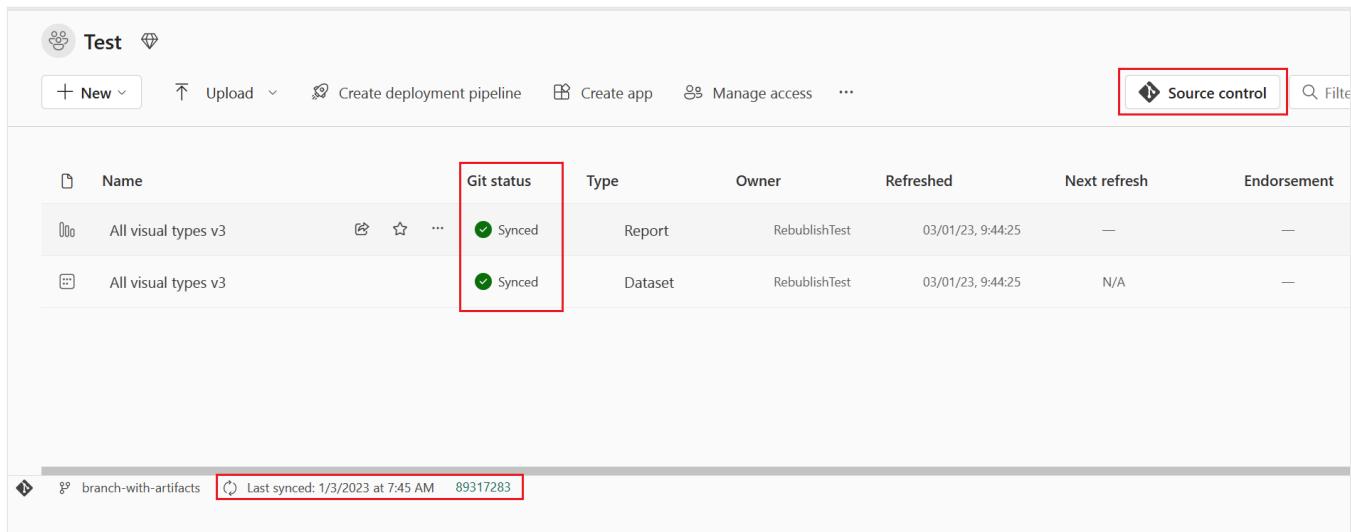
Enter name of folder

Connect and sync **Cancel**

Select **Connect and sync**.

During the initial sync, if either the workspace or Git branch is empty, content is copied from the nonempty location to the empty one. If both the workspace and Git branch have content, you're asked which direction the sync should go. For more information on this initial sync, see [Connect and sync](#).

After you connect, the Workspace displays information about source control that allows the user to view the connected branch, the status of each item in the branch and the time of the last sync.



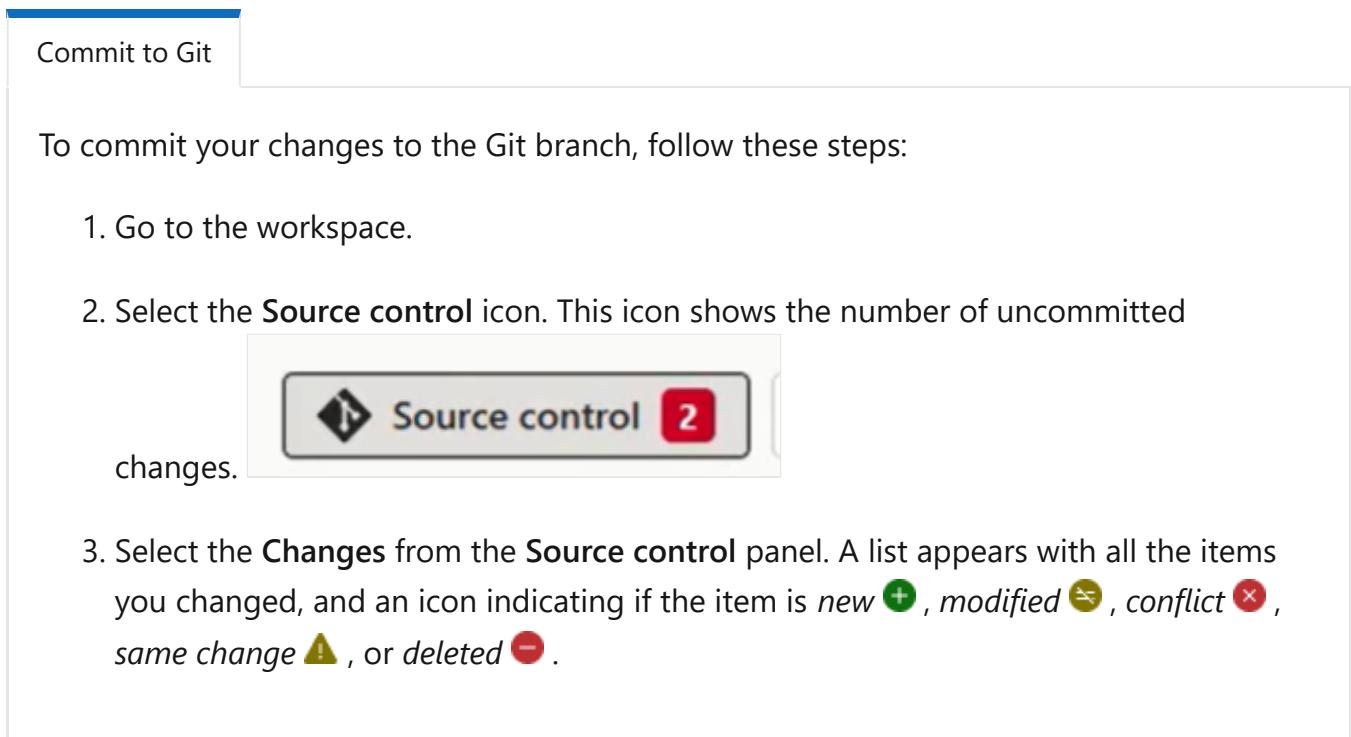
The screenshot shows the Power BI workspace interface. At the top, there's a navigation bar with options like '+ New', 'Upload', 'Create deployment pipeline', 'Create app', 'Manage access', and 'Source control'. The 'Source control' button is highlighted with a red box. Below the navigation bar is a table listing items from a Git branch. The columns are 'Name', 'Git status', 'Type', 'Owner', 'Refreshed', 'Next refresh', and 'Endorsement'. Two items are listed: 'All visual types v3' (Report, Synced) and 'All visual types v3' (Dataset, Synced). A red box highlights the 'Git status' column for both rows. At the bottom of the table, it says 'branch-with-artifacts' and 'Last synced: 1/3/2023 at 7:45 AM 89317283'.

To keep your workspace synced with the Git branch, [commit any changes](#) you make in the workspace to the Git branch, and [update your workspace](#) whenever anyone creates new commits to the Git branch.

Commit changes to git

Once you successfully connect to a Git folder, edit your workspace as usual. Any changes you save are saved in the workspace only. When you're ready, you can commit your changes to the Git branch, or you can undo the changes and revert to the previous status.

Read more about [commits](#).



The screenshot shows the Power BI workspace interface with a 'Commit to Git' button at the top left. Below it, a panel titled 'Source control' shows a count of 2 uncommitted changes. The panel lists three steps to commit changes:

1. Go to the workspace.
2. Select the **Source control** icon. This icon shows the number of uncommitted changes.
3. Select the **Changes** from the **Source control** panel. A list appears with all the items you changed, and an icon indicating if the item is *new* (+), *modified* (邈), *conflict* (X), *same change* (!), or *deleted* (-).

4. Select the items you want to commit. To select all items, check the top box.
5. Add a comment in the box. If you don't add a comment, a default message is added automatically.
6. Select Commit.

Changes 4 • Updates

Commit message

Add details before committing or we'll add for you by default.

<input checked="" type="checkbox"/>	Item	Status
<input checked="" type="checkbox"/>	Analysis	Synced
<input checked="" type="checkbox"/>	Analytics v2	Synced
<input checked="" type="checkbox"/>	Analysis	Synced
<input checked="" type="checkbox"/>	Analytics v2	Synced

Commit Undo

After the changes are committed, the items that were committed are removed from the list, and the workspace points to the new commit that it synced to.

Source control

X

demo123

Changes Updates



You don't have any changes

Any uncommitted changes from Git will be listed here.

After the commit is completed successfully, the status of the selected items changes from **Uncommitted** to **Synced**.

Update workspace from Git

Whenever anyone commits a new change to the connected Git branch, a notification appears in the relevant workspace. Use the **Source control** panel to pull the latest changes, merges, or reverts into the workspace and update live items. Changes to folders are also updated. Read more about [updating](#).

To update a workspace, follow these steps:

1. Go to the workspace.
2. Select the **Source control** icon.
3. Select **Updates** from the Source control panel. A list appears with all the items that were changed in the branch since the last update.
4. Select **Update all**.

Source control ⋮ ⌂ ×

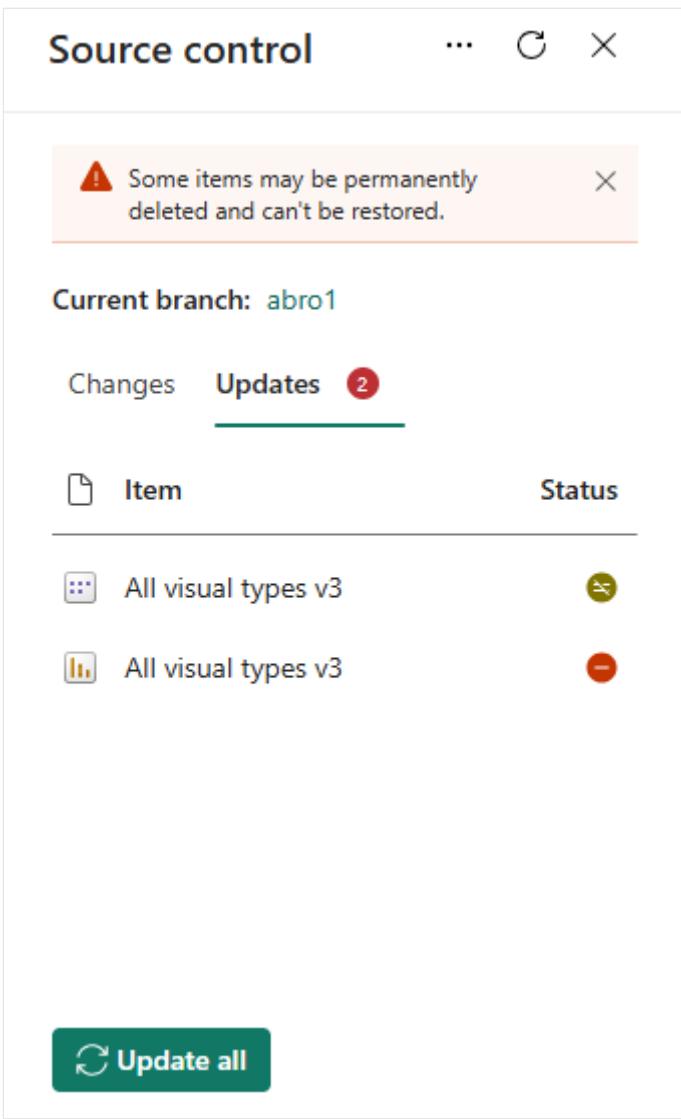
⚠ Some items may be permanently deleted and can't be restored. X

Current branch: abro1

Changes Updates 2

Item	Status
All visual types v3	green circle
All visual types v3	red circle

⟳ Update all

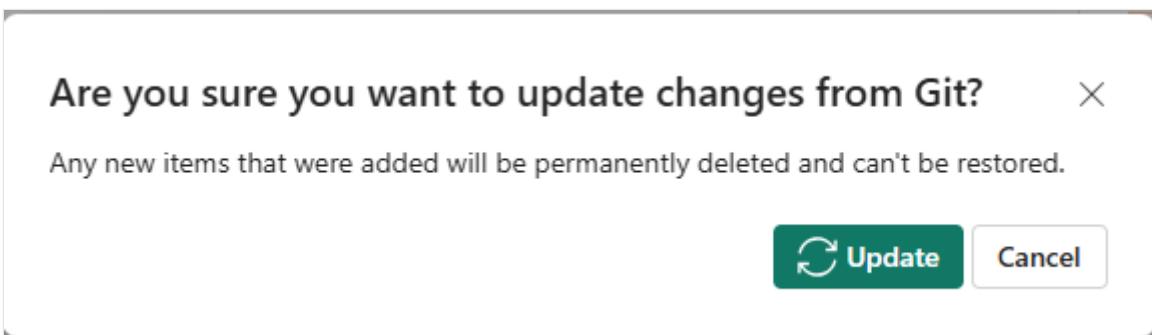


5. On the confirmation dialog, select **Update**.

Are you sure you want to update changes from Git? X

Any new items that were added will be permanently deleted and can't be restored.

⟳ Update **Cancel**



After it updates successfully, the list of items is removed, and the workspace points to the new workspace that it's synced to.

Source control

X

branch-with-artifacts

Changes 2 • Updates



You don't have any updates

Any unsynced updates from Git will be listed here.

After the update is completed successfully, the status of the items changes to **Synced**.

Disconnect a workspace from Git

Only a workspace admin can disconnect a workspace from a Git Repo. If you're not an admin, ask your admin for help with disconnecting. If you're an admin and want to disconnect your repo, follow these steps:

1. Go to **Workspace settings**
2. Select **Git integration**
3. Select **Disconnect workspace**
4. Select **Disconnect** again to confirm.

Permissions

The actions you can take on a workspace depend on the permissions you have in both the workspace and the Git repo. For a more detailed discussion of permissions, see [Permissions](#).

Considerations and limitations

General Git integration limitations

- The [authentication method](#) in Fabric must be at least as strong as the authentication method for Git. For example, if Git requires multifactor authentication, Fabric needs to require multifactor authentication as well.
- Power BI Datasets connected to Analysis Services aren't supported at this time.
- If you use a workspace identity in one artifact and commit it to Git, it can be updated (back to a fabric workspace) only in a workspace connected to the same identity. Be careful, as this also affects features like branch out.
- Submodules aren't supported.
- Sovereign clouds aren't supported.
- If your workspace contains hundreds of items, consider splitting it into smaller sets of artifacts. Each set should be placed in a separate workspace and linked to a different Git branch, or connected to a single branch organized into different folders.

Azure DevOps limitations

- Azure DevOps isn't supported if [Enable IP Conditional Access policy validation](#) is enabled.
- If the workspace and Git repo are in two different geographical regions, the tenant admin must enable [cross-geo exports](#).
- If your organization configured [conditional access](#), make sure the **Power BI Service** has the same [conditions set](#) for authentication to function as expected.
- The following commit size limit is applied:
 - 25 MB using the Azure DevOps connector with Service Principal.
 - 125 MB using the default single sign-on (SSO) Microsoft Entra ID account and Azure DevOps connector with User Principal.

GitHub Enterprise limitations

Some GitHub Enterprise versions and settings aren't supported. For example:

- GitHub Enterprise Cloud with data residency (ghe.com)
- GitHub Enterprise Server with a custom domain is not supported, even if the instance is publicly accessible
- Github Enterprise Server hosted on a private network
- IP allowlist

Azure DevOps to GitHub Enterprise migration consideration

If your team uses Fabric Git Integration and is evaluating a migration from Azure DevOps to GitHub Enterprise, it's recommended to run validation tests to ensure Git Integration functionality remains unaffected. Fabric Git Integration relies on the underlying Git provider APIs, which differ in capabilities and limitations between Azure DevOps and GitHub Enterprise, as described above.

Workspace limitations

- Only the workspace admin can manage the connections to the [Git Repo](#) such as connecting, disconnecting, or adding a branch.
Once connected, anyone with [permission](#) can work in the workspace.
- Workspaces with template apps installed can't be connected to Git.
- [MyWorkspace](#) can't connect to a Git provider.

Branch and folder limitations

- Maximum length of branch name is 244 characters.
- Maximum length of full path for file names is 250 characters. Longer names fail.
- Maximum file size is 25 MB.
- Folder structure is maintained up to 10 levels deep.
- Downloading a report/dataset as [.pbix](#) from the service after deploying them with Git integration is not recommended, as the results are unreliable. We recommend using PowerBI Desktop to download reports/datasets as [.pbix](#).
- If the item's display name has any of these characteristics, The Git folder is renamed to the logical ID (Guid) and type:
 - Has more than 256 characters
 - Ends with a [.](#) or a space
 - Contains any forbidden characters as described in [directory name limitations](#)
- When you connect a workspace that has folders to Git, you need to commit changes to the Git repo if that [folder structure](#) is different.

Directory name limitations

- The name of the directory that connects to the Git repository has the following naming restrictions:
 - The directory name can't begin or end with a space or tab.
 - The directory name can't contain any of the following characters: " / : < > \ * ?



- The item folder (the folder that contains the item files) can't contain any of the following characters: " : < > \ * ? | . If you rename the folder to something that includes one of these characters, Git can't connect or sync with the workspace and an error occurs.

Branching out limitations

- Branch out requires permissions listed in [permissions table](#).
- There must be an available capacity for this action.
- All [workspace](#) and [branch naming limitations](#) apply when branching out to a new workspace.
- Only [Git supported items](#) are available in the new workspace.
- The related branches list only shows branches and workspaces you have permission to view.
- [Git integration](#) must be enabled.
- When branching out, a new branch is created and the settings from the original branch aren't copied. Adjust any settings or definitions to ensure that the new meets your organization's policies.
- When branching out to an existing workspace:
 - The target workspace must support a Git connection.
 - The user must be an admin of the target workspace.
 - The target workspace must have capacity.
 - The workspace can't have template apps.
- Note that when you branch out to a workspace, any items that aren't saved to Git can get lost. We recommend that you [commit](#) any items you want to keep before branching out.

Sync and commit limitations

- You can only sync in one direction at a time. You can't commit and update at the same time.
- Sensitivity labels aren't supported and exporting items with sensitivity labels might be disabled. To commit items that have sensitivity labels without the sensitivity label, [ask your administrator](#) for help.
- Works with [limited items](#). Unsupported items in the folder are ignored.
- Duplicating names isn't allowed. Even if Power BI allows name duplication, the update, commit, or undo action fails.
- B2B isn't supported.
- [Conflict resolution](#) is partially done in Git.
- During the *Commit to Git* process, the Fabric service deletes files *inside the item folder* that aren't part of the item definition. Unrelated files not in an item folder aren't deleted.

- After you commit changes, you might notice some unexpected changes to the item that you didn't make. These changes are semantically insignificant and can happen for several reasons. For example:
 - Manually changing the item definition file. These changes are valid, but might be different than if done through the editors. For example, if you rename a semantic model column in Git and import this change to the workspace, the next time you commit changes to the semantic model, the *bim* file will register as changed and the modified column pushed to the back of the `columns` array. This is because the AS engine that generates the *bim* files pushes renamed columns to the end of the array. This change doesn't affect the way the item operates.
 - Committing a file that uses *CRLF* line breaks. The service uses *LF* (line feed) line breaks. If you had item files in the Git repo with *CRLF* line breaks, when you commit from the service these files are changed to *LF*. For example, if you open a report in desktop, save the project file (*.pbip*) and upload it to Git using *CRLF*.
- Refreshing a semantic model using the [Enhanced refresh API](#) causes a Git diff after each refresh.

Related content

- [Understand the Git integration process](#)
- [Manage Git branches](#)
- [Git integration best practices](#)

Last updated on 12/16/2025

Basic concepts in Git integration

This article explains basic Git concepts and the process of integrating Git with your Microsoft Fabric workspace.

Permissions

- Your organization's administrator must [enable Git integration](#).
- The tenant admin must [enable cross-geo export](#) if the workspace and Azure repo are in two different regions. This restriction doesn't apply to GitHub.
- The permissions you have in both the workspace and Git, as listed in the next sections, determine the actions you can take.

Required Git permissions for popular actions

The following list shows what different workspace roles can do depending on their permissions in their Git repo:

- **Admin:** Can perform any operation on the workspace, limited only by their Git role.
- **Member/Contributor:** Once they connect to a workspace, a member/contributor can commit and update changes, depending on their Git role. For actions related to the workspace connection (for example, connect, disconnect, or switch branches) seek help from an Admin.
- **Viewer:** Can't perform any actions. The viewer can't see any Git related information in the workspace.

Required Fabric permissions for popular actions

Workspace roles

The following table describes the permissions needed in the Fabric workspace to perform various common operations:

 [Expand table](#)

Operation	Workspace role
Connect workspace to Git repo	Admin
Sync workspace with Git repo	Admin
Disconnect workspace from Git repo	Admin

Operation	Workspace role
Switch branch in the workspace (or any change in connection setting)	Admin
View Git connection details	Admin, Member, Contributor
See workspace 'Git status'	Admin, Member, Contributor
Update from Git	<p>All of the following roles:</p> <p>Contributor in the workspace (WRITE permission on all items)</p> <p>Owner of the item (if the tenant switch blocks updates for nonowners)</p> <p>BUILD on external dependencies (where applicable)</p>
Commit workspace changes to Git	<p>All of the following roles:</p> <p>Contributor in the workspace (WRITE permission on all items)</p> <p>Owner of the item (if the tenant switch blocks updates for nonowners)</p> <p>BUILD on external dependencies (where applicable)</p>
Create new Git branch from within Fabric	Admin
Branch out to another workspace	Admin, Member, Contributor

Git roles

The following table describes the Git permissions needed to perform various common operations:

Azure Repos

[] Expand table

Operation	Git permissions
Connect workspace to Git repo	Read=Allow
Sync workspace with Git repo	Read=Allow

Operation	Git permissions
Disconnect workspace from Git repo	No permissions are needed
Switch branch in the workspace (or any change in connection setting)	Read=Allow (in target repo/directory/branch)
View Git connection details	Read or None
See workspace 'Git status'	Read=Allow
Update from Git	Read=Allow
Commit workspace changes to Git	Read=Allow Contribute=Allow branch policy should allow direct commit
Create new Git branch from within Fabric	Role=Write Create branch=Allow
Branch out to another workspace	Read=Allow Create branch=Allow

Connect and sync

Only a workspace admin can connect a workspace to a Git Repos, but once connected, anyone with permissions can work in the workspace. If you're not an admin, ask your admin for help with connecting.

When you [connect a workspace to Git](#), Fabric syncs between the two locations so they have the same content. During this initial sync, if either the workspace or Git branch is empty while the other has content, the content is copied from the nonempty location to the empty one. If both the workspace and Git branch have content, you have to decide which direction the sync should go.

- If you commit your workspace to the Git branch, all supported workspace content is exported to Git and overwrites the current Git content.
- If you update the workspace with the Git content, the workspace content is overwritten, and you lose your workspace content. Since a Git branch can always be restored to a previous stage while a workspace can't, if you choose this option, you're asked to confirm.

What content do you want to sync?

X

 Some items may be permanently deleted and can't be restored. [Learn more](#) ↗

You have existing content in both Git and this workspace. Select the content you want to keep and sync to continue. [Learn more](#)

Sync content from this workspace into Git

Sync content from Git into this workspace

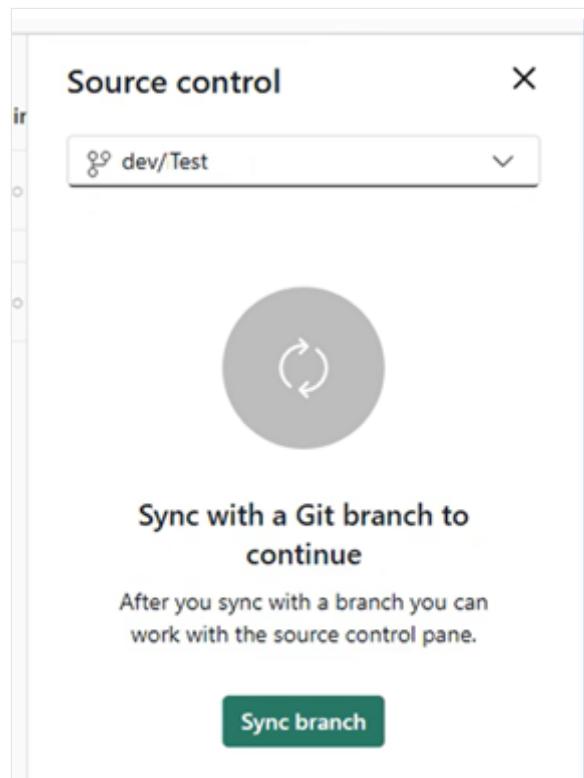
 This will override all items in this workspace

I understand workspace items may be deleted and can't be restored.

Sync

Cancel

If you don't select which content to sync, you can't continue to work.

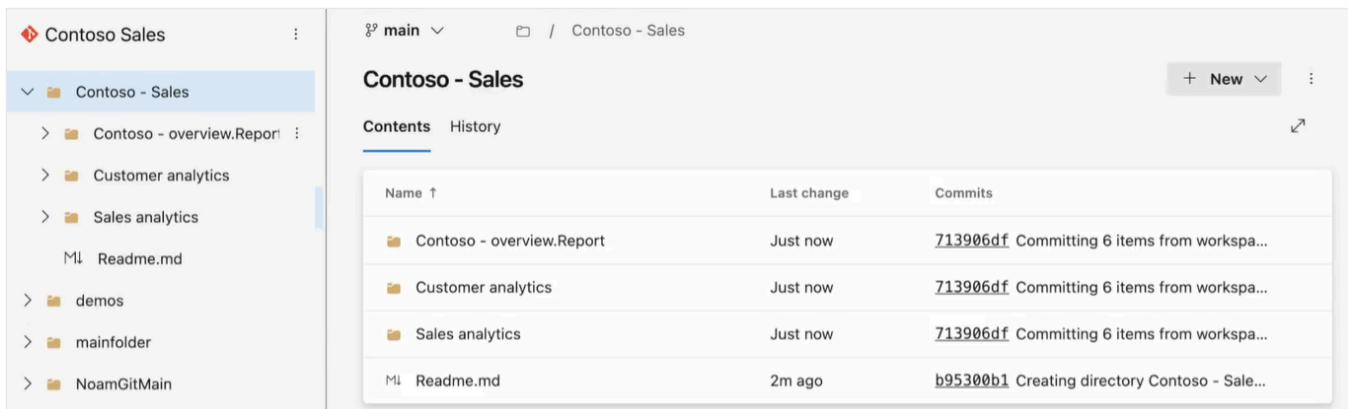


Folders

When connected and synced, the workspace structure is mirrored in the Git repository, including folders structure. Workspace items in folders are exported to folders with the same name in the Git repo. Conversely, items in Git folders are imported to folders with the same name in the workspace.

⚠ Note

Since folder structure is retained, if your workspace has folders and the connected Git folder doesn't yet have subfolders, they're considered to be different. You get an *uncommitted changes* status in the source control panel and you need to commit the changes to Git before updating the workspace. If you update first, the Git folder structure **overwrites the workspace** folder structure. For more information, see [Handling folder changes safely](#).



- Empty folders aren't copied to Git. When you create or move items to a folder, the folder is created in Git.
- Empty folders in Git are deleted automatically.
- Empty folders in the workspace aren't deleted automatically even if all items are moved to different folders.
- Folder structure is retained up to 10 levels deep.

Handling folder changes safely

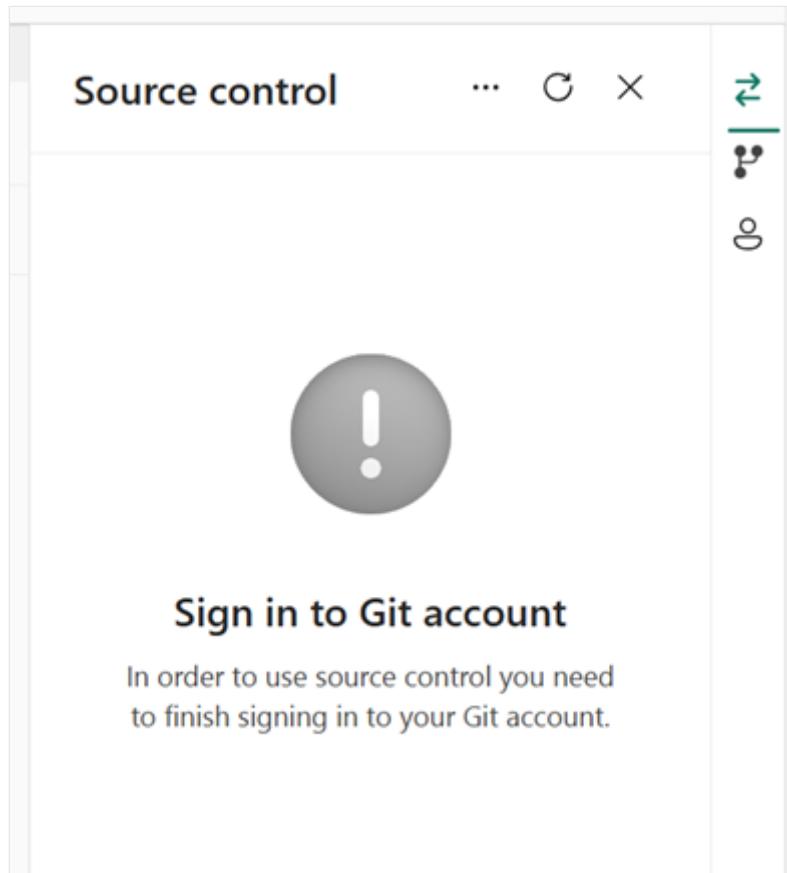
If your workspace has folders and the connected Git folder doesn't yet have subfolders, they're considered to be different because the folder structure is different. When you connect a workspace that has folders to Git, you get an *uncommitted changes* status in the source control panel and you need to commit the changes to Git before updating the workspace.

If you can't make changes to the connected branch directly, due to branch policy or permissions, we recommend using the *Checkout Branch* option:

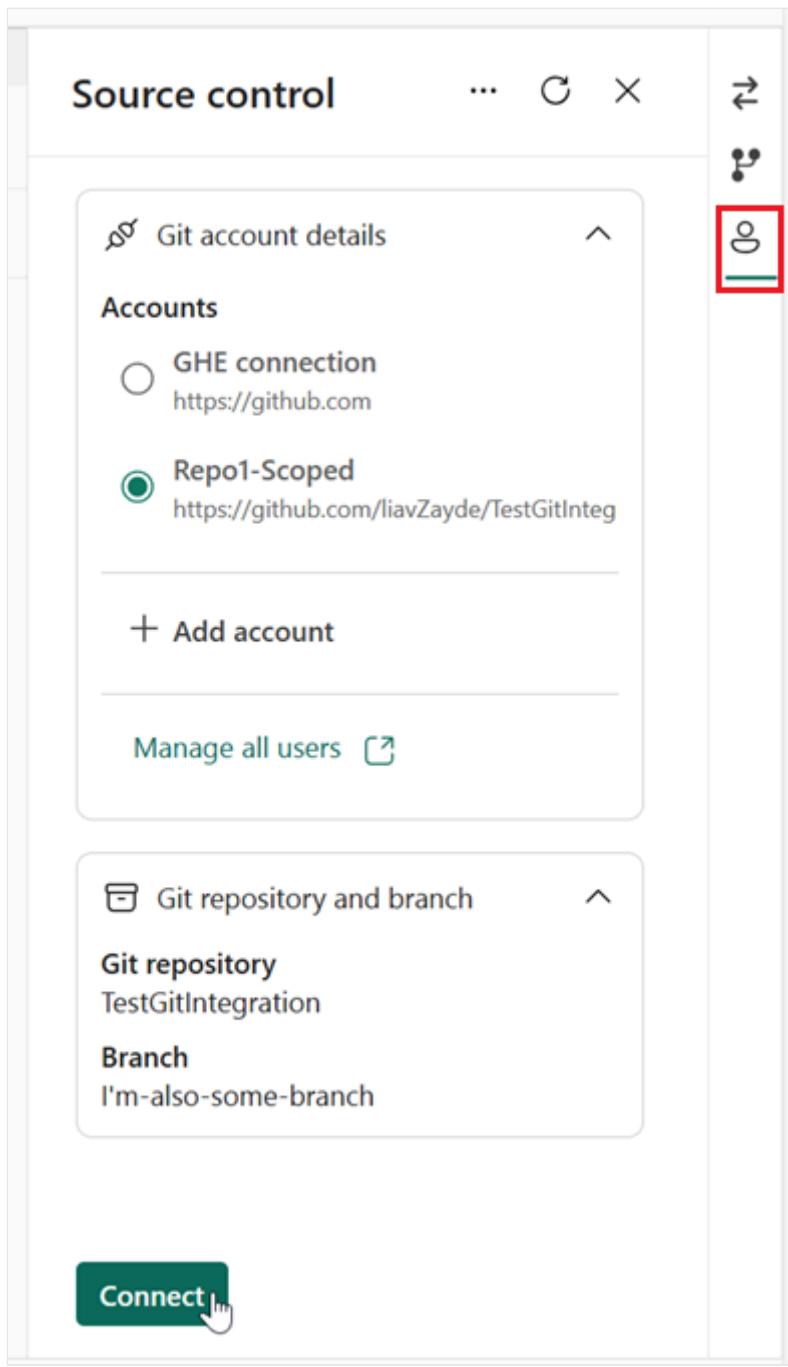
1. **Checkout a New Branch:** Use the checkout branch feature to create a branch with the updated state of your Fabric workspace.
2. **Commit Folder Changes:** Any workspace folder changes can then be committed to this new branch.
3. **Merge Changes:** Use your regular pull request (PR) and merge processes to integrate these updates back into the original branch.

Connect to a shared workspace

If you try connecting to a workspace that's already [connected to Git](#), you might get the following message:

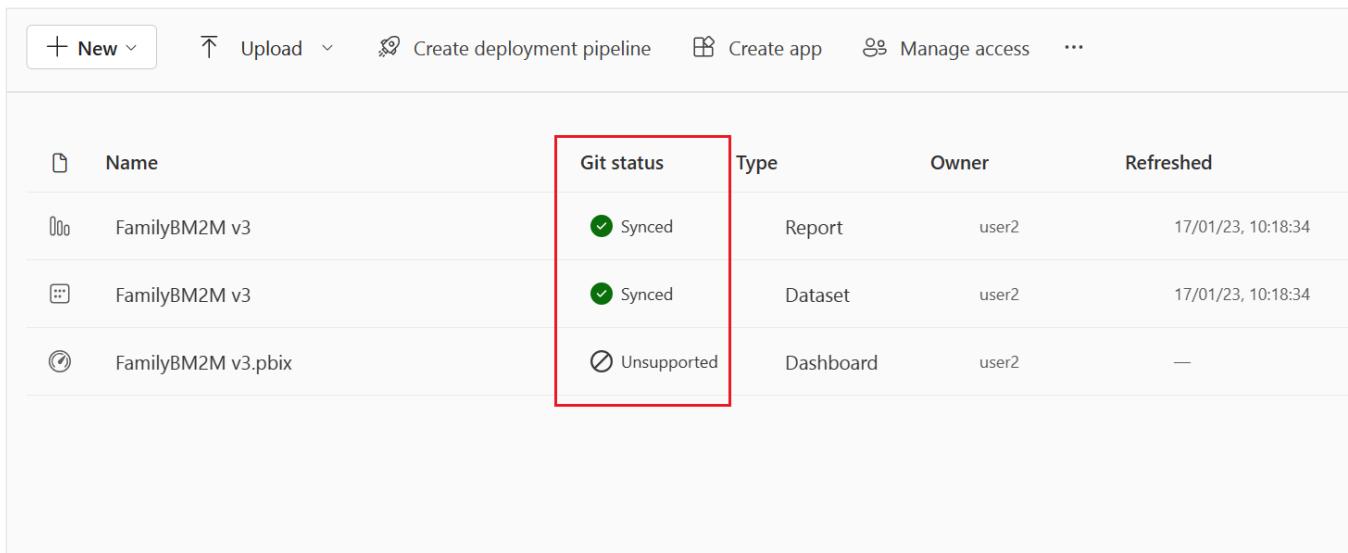


Go to the **Accounts** tab on the right side of the Source control panel, choose an account, and connect to it.



Git status

After you connect, the workspace displays a *Git status* column that indicates the sync state of each item in the workspace in relation to the items in the remote branch.



Name	Git status	Type	Owner	Refreshed
FamilyBM2M v3	Synced	Report	user2	17/01/23, 10:18:34
FamilyBM2M v3	Synced	Dataset	user2	17/01/23, 10:18:34
FamilyBM2M v3.pbix	Unsupported	Dashboard	user2	—

Each item has one of the following statuses:

- Synced (the item is the same in the workspace and Git branch)
- Conflict (the item was changed in both the workspace and Git branch)
- Unsupported item
- Uncommitted changes in the workspace
- Update required from Git
- Item is identical in both places but needs to be updated to the last commit

Sync information

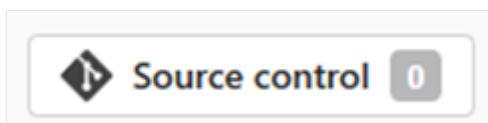
As long as you're connected, the following information appears at the bottom of your screen:

- Connected branch
- Time of last sync
- Link to the last commit that the workspace is synced to



Source control pane

On top of the screen is the **Source control** icon. It shows the number of items that are different in the workspace and Git branch. When changes are made either to the workspace or the Git branch, the number is updated. When the workspace is synced with the Git branch, the Source control icon displays a 0.



Select the Source control icon to open the **Source control** panel.

The source control pane has three tabs on the side:

- [Commits and updates](#)
- [Branches](#)
- [Account details](#)

Commits and updates

When changes are made either to the workspace or the Git branch, the source control icon shows the number of items that are different. Select the source control icon to open the Source control panel.

The **Commit and update** panel has two sections.

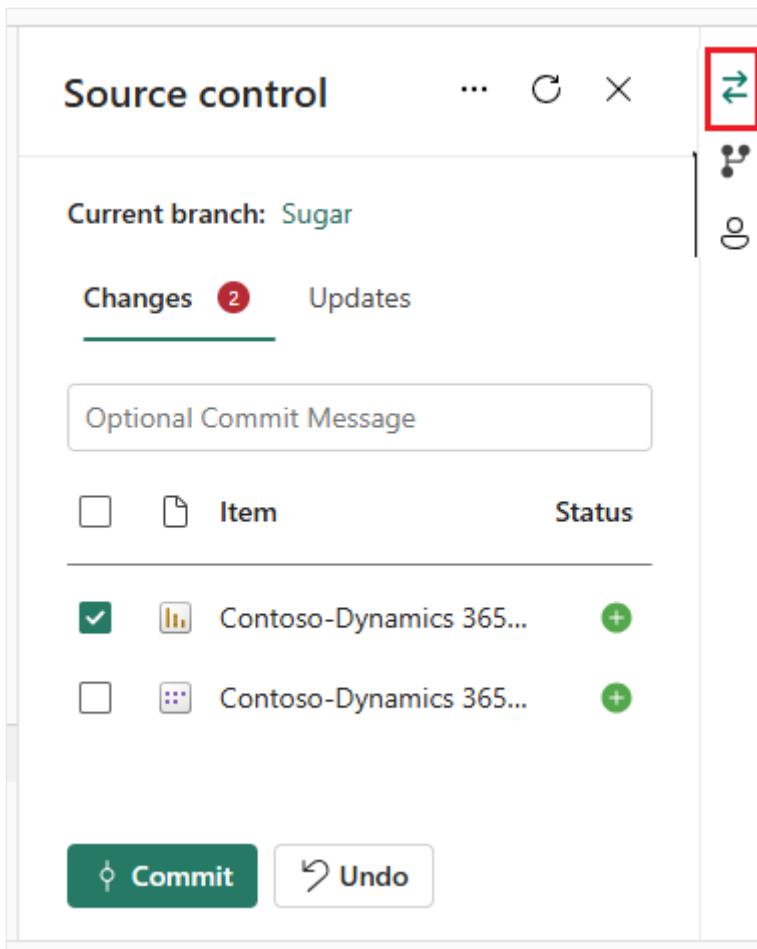
Changes shows the number of items that were changed in the workspace and need to be committed to Git.

Updates shows the number of items that were modified in the Git branch and need to be updated to the workspace.

In each section, the changed items are listed with an icon indicating the status:

-  new
-  modified
-  deleted
-  conflict
-  same-changes

The Refresh button  on top of the panel updates the list of changes and updates.



Commit

- Items in the workspace that were changed are listed in the *Changes* section. When there's more than one changed item, you can select which items to commit to the Git branch.
- If there were updates made to the Git branch, commits are disabled until you update your workspace.

Update

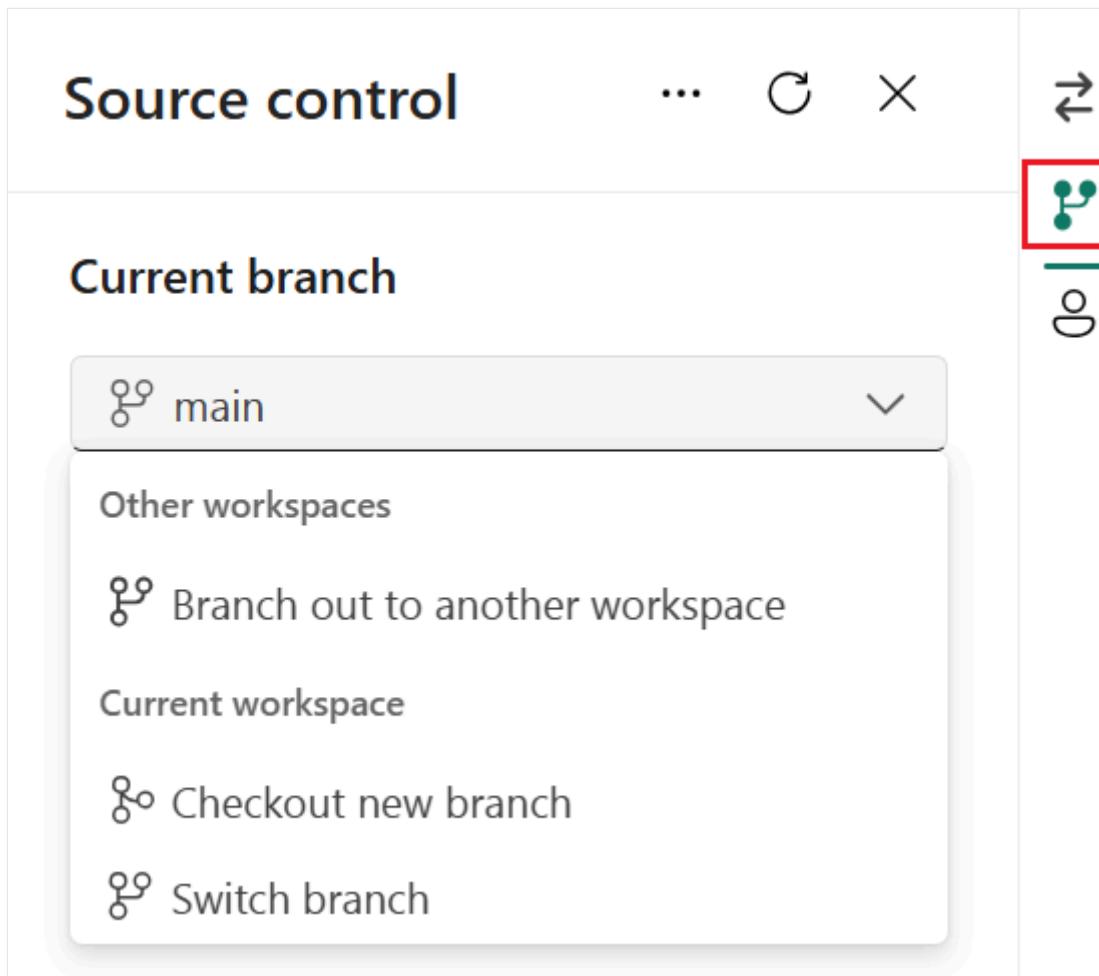
- Unlike *commit* and *undo*, the *Update* command always updates the entire branch and syncs to the most recent commit. You can't select specific items to update.
- If changes were made in the workspace and in the Git branch *on the same item*, updates are disabled until the [conflict is resolved](#).

Read more about how to [commit](#) and [update](#). Read more about the update process and how to [resolve conflicts](#).

Branches

The *Branches* tab of the Source control panel enables you to manage your branches and perform branch related actions. It has two main sections:

- Actions you can take on the current branch:
 - [Branch out to another workspace](#) (contributor and above): Creates a new workspace, or switches to an existing workspace based on the last commit to the current workspace. It then connects to the target workspace and branch.
 - [Checkout new branch](#) (must be workspace admin): Creates a new branch based on the last synced commit in the workspace and changes the Git connection in the current workspace. It doesn't change the workspace content.
 - [Switch branch](#) (must be workspace admin): Syncs the workspace with another new or existing branch and overrides all items in the workspace with the content of the selected branch.

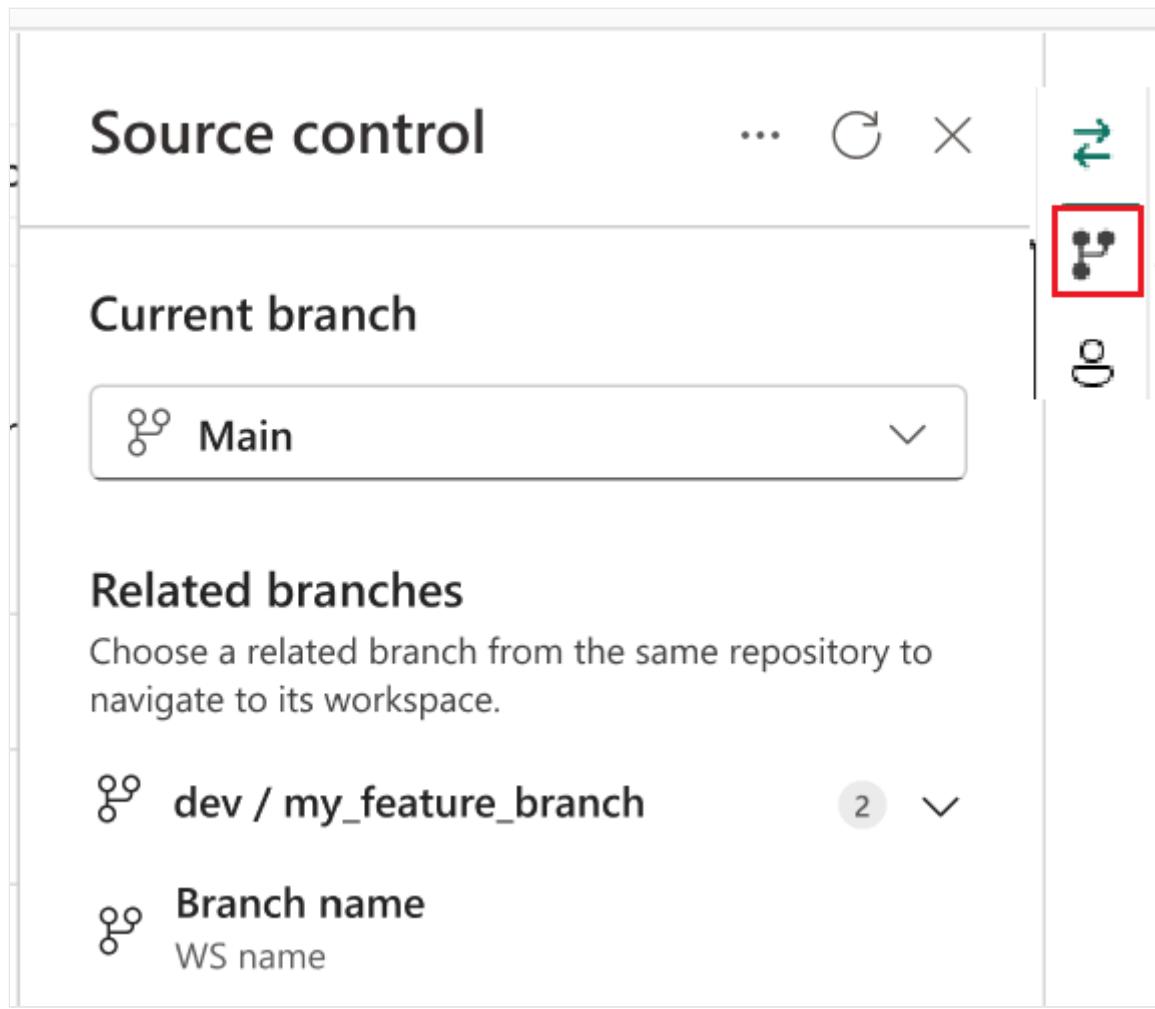


- Related branches.

The *Branches* tab also has a list of related workspaces you can select and switch to. A related workspace is one with the same connection properties as the current branch, such as the same organization, project, repository, and git folder.

This feature allows you to navigate to workspaces connected to other branches related to the context of your current work, without having to look for them in your list of Fabric workspaces.

To open the relevant workspace, select item in the list.



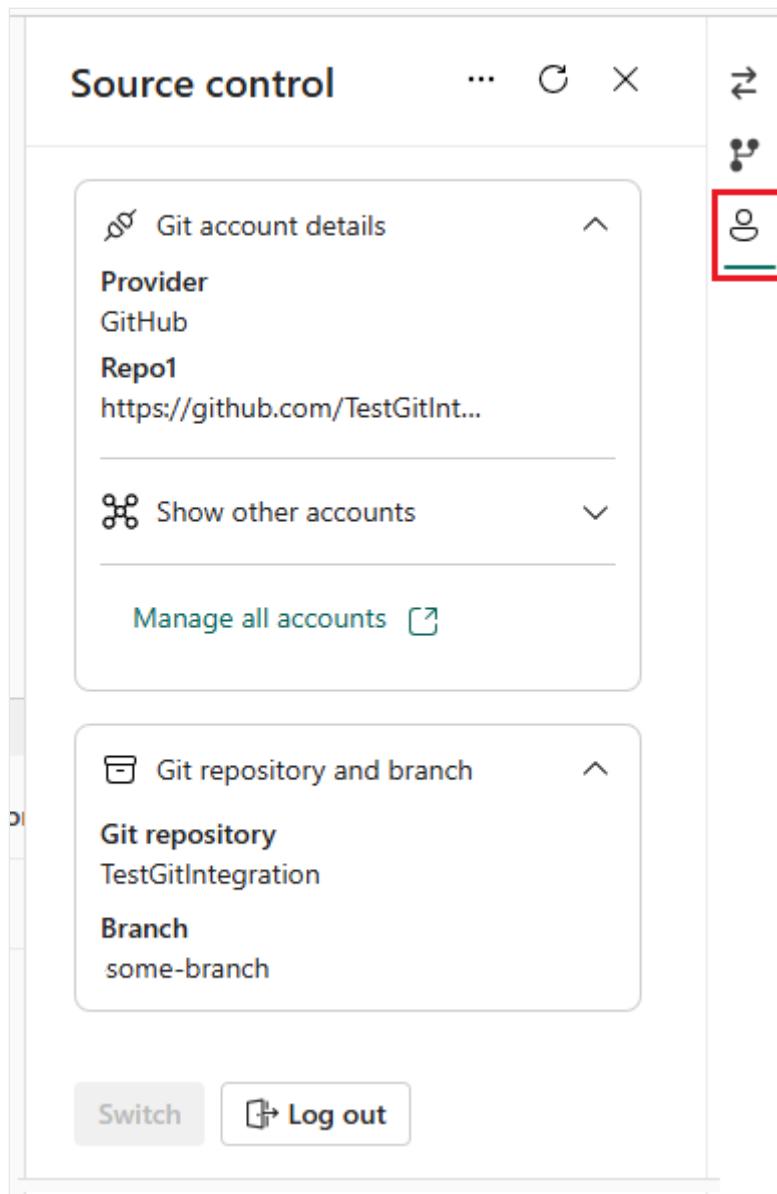
For more information, see [Branching out limitations](#).

Account details

The Account details tab shows details of the GitHub account that the user is connected to. It has two sections. The top section shows the Git provider and the account name. The bottom section shows the repository and branch that the workspace is connected to. Currently, this tab is only available for workspaces connected to GitHub.

GitHub account details include:

- Git account details
 - Provider
 - Account name
- Git repository
- Branch



Considerations and limitations

General Git integration limitations

- The [authentication method](#) in Fabric must be at least as strong as the authentication method for Git. For example, if Git requires multifactor authentication, Fabric needs to require multifactor authentication as well.
- Power BI Datasets connected to Analysis Services aren't supported at this time.
- If you use a workspace identity in one artifact and commit it to Git, it can be updated (back to a fabric workspace) only in a workspace connected to the same identity. Be careful, as this also affects features like branch out.
- Submodules aren't supported.
- Sovereign clouds aren't supported.
- If your workspace contains hundreds of items, consider splitting it into smaller sets of artifacts. Each set should be placed in a separate workspace and linked to a different Git

branch, or connected to a single branch organized into different folders.

Azure DevOps limitations

- Azure DevOps isn't supported if [Enable IP Conditional Access policy validation](#) is enabled.
- If the workspace and Git repo are in two different geographical regions, the tenant admin must enable [cross-geo exports](#).
- If your organization configured [conditional access](#), make sure the **Power BI Service** has the same [conditions set](#) for authentication to function as expected.
- The following commit size limit is applied:
 - 25 MB using the Azure DevOps connector with Service Principal.
 - 125 MB using the default single sign-on (SSO) Microsoft Entra ID account and Azure DevOps connector with User Principal.

GitHub Enterprise limitations

Some GitHub Enterprise versions and settings aren't supported. For example:

- GitHub Enterprise Cloud with data residency (ghe.com)
- GitHub Enterprise Server with a custom domain is not supported, even if the instance is publicly accessible
- GitHub Enterprise Server hosted on a private network
- IP allowlist

Azure DevOps to GitHub Enterprise migration consideration

If your team uses Fabric Git Integration and is evaluating a migration from Azure DevOps to GitHub Enterprise, it's recommended to run validation tests to ensure Git Integration functionality remains unaffected. Fabric Git Integration relies on the underlying Git provider APIs, which differ in capabilities and limitations between Azure DevOps and GitHub Enterprise, as described above.

Workspace limitations

- Only the workspace admin can manage the connections to the [Git Repo](#) such as connecting, disconnecting, or adding a branch.
Once connected, anyone with [permission](#) can work in the workspace.
- Workspaces with template apps installed can't be connected to Git.
- [MyWorkspace](#) can't connect to a Git provider.

Branch and folder limitations

- Maximum length of branch name is 244 characters.
- Maximum length of full path for file names is 250 characters. Longer names fail.
- Maximum file size is 25 MB.
- Folder structure is maintained up to 10 levels deep.
- Downloading a report/dataset as `.pbix` from the service after deploying them with Git integration is not recommended, as the results are unreliable. We recommend using PowerBI Desktop to download reports/datasets as `.pbix`.
- If the item's display name has any of these characteristics, The Git folder is renamed to the logical ID (Guid) and type:
 - Has more than 256 characters
 - Ends with a `.` or a space
 - Contains any forbidden characters as described in [directory name limitations](#)
- When you connect a workspace that has folders to Git, you need to commit changes to the Git repo if that [folder structure](#) is different.

Directory name limitations

- The name of the directory that connects to the Git repository has the following naming restrictions:
 - The directory name can't begin or end with a space or tab.
 - The directory name can't contain any of the following characters: `" / : < > \ * ? |`
- The item folder (the folder that contains the item files) can't contain any of the following characters: `" : < > \ * ? |`. If you rename the folder to something that includes one of these characters, Git can't connect or sync with the workspace and an error occurs.

Branching out limitations

- Branch out requires permissions listed in [permissions table](#).
- There must be an available capacity for this action.
- All [workspace](#) and [branch naming limitations](#) apply when branching out to a new workspace.
- Only [Git supported items](#) are available in the new workspace.
- The related branches list only shows branches and workspaces you have permission to view.
- [Git integration](#) must be enabled.
- When branching out, a new branch is created and the settings from the original branch aren't copied. Adjust any settings or definitions to ensure that the new meets your

organization's policies.

- When branching out to an existing workspace:
 - The target workspace must support a Git connection.
 - The user must be an admin of the target workspace.
 - The target workspace must have capacity.
 - The workspace can't have template apps.
- Note that when you branch out to a workspace, any items that aren't saved to Git can get lost. We recommend that you [commit](#) any items you want to keep before branching out.

Sync and commit limitations

- You can only sync in one direction at a time. You can't commit and update at the same time.
- Sensitivity labels aren't supported and exporting items with sensitivity labels might be disabled. To commit items that have sensitivity labels without the sensitivity label, [ask your administrator](#) for help.
- Works with [limited items](#). Unsupported items in the folder are ignored.
- Duplicating names isn't allowed. Even if Power BI allows name duplication, the update, commit, or undo action fails.
- B2B isn't supported.
- [Conflict resolution](#) is partially done in Git.
- During the *Commit to Git* process, the Fabric service deletes files *inside the item folder* that aren't part of the item definition. Unrelated files not in an item folder aren't deleted.
- After you commit changes, you might notice some unexpected changes to the item that you didn't make. These changes are semantically insignificant and can happen for several reasons. For example:
 - Manually changing the item definition file. These changes are valid, but might be different than if done through the editors. For example, if you rename a semantic model column in Git and import this change to the workspace, the next time you commit changes to the semantic model, the *bim* file will register as changed and the modified column pushed to the back of the `columns` array. This is because the AS engine that generates the *bim* files pushes renamed columns to the end of the array. This change doesn't affect the way the item operates.
 - Committing a file that uses *CRLF* line breaks. The service uses *LF* (line feed) line breaks. If you had item files in the Git repo with *CRLF* line breaks, when you commit from the service these files are changed to *LF*. For example, if you open a report in desktop, save the project file (*.pbip*) and upload it to Git using *CRLF*.
- Refreshing a semantic model using the [Enhanced refresh API](#) causes a Git diff after each refresh.

Related content

- [Manage branches](#)
- [Resolve errors and conflicts](#)

Last updated on 12/15/2025

Manage branches in Microsoft Fabric workspaces

The goal of this article is to present Fabric developers with different options for building CI/CD processes in Fabric, based on common customer scenarios. This article focuses more on the *continuous integration (CI)* part of the CI/CD process. For a discussion of the continuous delivery (CD) part, see [manage deployment pipelines](#).

This article outlines a few distinct integration options, but many organizations use a combination of them.

Prerequisites

To integrate Git with your Microsoft Fabric workspace, you need to set up the following prerequisites for both Fabric and Git.

Fabric prerequisites

To access the Git integration feature, you need a [Fabric capacity](#). A Fabric capacity is required to use all supported Fabric items. If you don't have one yet, [sign up for a free trial](#). Customers that already have a [Power BI Premium capacity](#), can use that capacity, but keep in mind that [certain Power BI SKUs only support Power BI items](#).

In addition, the following [tenant switches](#) must be enabled from the Admin portal:

- [Users can create Fabric items](#)
- [Users can synchronize workspace items with their Git repositories](#)
- [Create workspaces](#) (only if you want to branch out to a new workspace.)
- [Users can synchronize workspace items with GitHub repositories](#): For GitHub users only

These switches can be enabled by the tenant admin, capacity admin, or workspace admin, depending on your [organization's settings](#).

Git prerequisites

Git integration is currently supported for Azure DevOps and GitHub. To use Git integration with your Fabric workspace, you need the following in either Azure DevOps or GitHub:

Azure DevOps

- An Active **Azure DevOps account** registered to same Fabric user (supported even if Azure DevOps organization reside in a different tenant than Fabric tenant). [Create a free account ↗](#).
- Access to an existing repository.

Development process

The Fabric workspace is a shared environment that accesses live items. Any changes made directly in the workspace override and affect all other workspace users. Therefore, Git best practice is for developers to work in isolation outside of the shared workspaces. There are two ways for a developer to work in their own protected workspace.

- [Develop using client tools](#), such as [Power BI Desktop ↗](#) for reports and semantic models, or [VS Code ↗](#) for Notebooks.
- [Develop in a separate Fabric workspace](#). Each developer has their own workspace where they connect their own separate branch, sync the content into that workspace, and then commit back to the branch.

To work with branches using Git integration, first connect the shared development team's workspace to a single shared branch. For example, if your team uses one shared workspace, connect it to the *main* branch in your team's repository, and sync between the workspace and the repo. If your team's workflow has multiple shared branches like *Dev/Test/Prod* branches, each branch can be connected to a different workspace.

Then, each developer can choose the isolated environment in which to work.

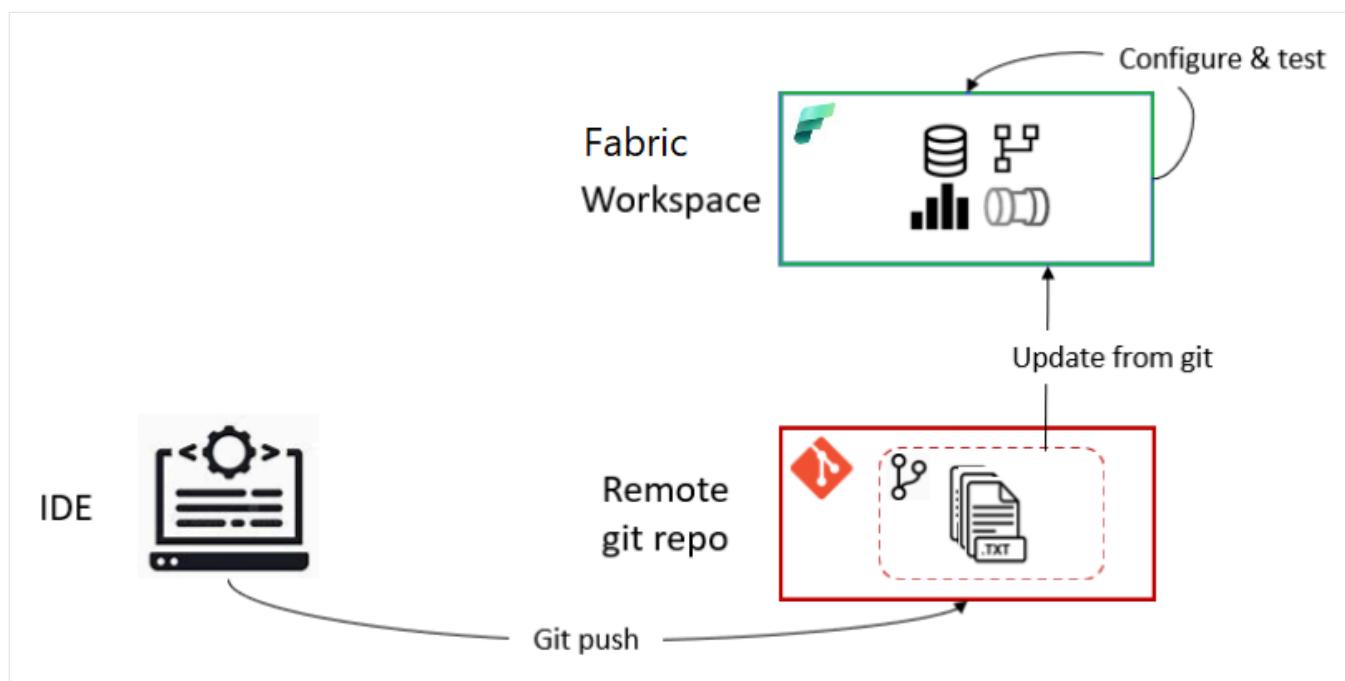
Scenario 1 - Develop using client tools

If the items you're developing are available in other tools, you can work on those items directly in the client tool. Not all items are available in every tool. Items that are only available in Fabric need to be developed in Fabric.

The workflow for developers using a client tool like Power BI Desktop should look something like this:

1. [Clone](#) the repo onto a local machine. (You only need to do this step once.)
2. Open the project in Power BI Desktop using the local copy of the *PBIProj*.
3. Make changes and save the updated files locally. [Commit](#) to the local repo.

4. When ready, [push](#) the branch and commits to the remote repo.
5. Test the changes against other items or against more data. To test the changes, connect the new branch to a separate workspace, and upload the semantic model and reports using the *update all* button in the source control panel. Do any tests or configuration changes there before merging into the *main* branch.
If no tests are required in the workspace, the developer can merge changes directly into the *main* branch, without the need for another workspace.
6. Once the changes are merged, the shared team's workspace is prompted to accept the new commit. The changes are updated into the shared workspace and everyone can see the changes to those semantic models and reports.



For a specific guidance on how to use the new Power BI Desktop file format in git, see [Source code format](#).

Scenario 2 - Develop using another workspace

For a developer who works in the web, the flow would be as follows:

1. From the *Branches* tab of the **Source control** menu, select **Branch out to another workspace**.

Source control

... C X



Current branch

main ▾

Other workspaces

Branch out to another workspace

Current workspace

Checkout new branch

Switch branch

2. Specify if you want to create a new workspace or switch to an existing one. Specify the names of the new branch and workspace, or select the existing workspace from the dropdown list. You will see the following screenshot when creating a new workspace.

! Note

When you branch out to a workspace, any items that aren't saved to Git can get lost. We recommend that you commit any items you want to keep before branching out.

Branch out to another workspace

X

⚠️ Uncommitted changes Uncommitted changes in the current workspace will not be included in the new workspace. [Learn more](#) ↗

Create a copy of your current branch, connect it to a new workspace, and make changes without affecting your current workspace. [Learn more](#)

Branch

Name *

Add new branch name

Based on ⓘ

main

Workspace ⓘ

New

Add new workspace name

Existing

Select existing workspace

[Why can't I see all my workspaces?](#)

Branch out

Cancel

ⓘ Important

When you branch out to an existing workspace some items may be deleted.

For an existing workspace, you will see the screenshot below which warns that connecting to an existing workspace may result in some items being deleted.

Branch out to workspace

X

⚠️ Uncommitted changes will not be included in the new branch's workspace. If you connect to an existing workspace, some items in that workspace may be deleted and can't be restored. [Learn more](#) ↗

Create a copy of your current branch, connect it to another workspace, and make changes without affecting your current workspace. [Learn more](#)

Branch

Name *

new_branch

Based on ⓘ

abro_test

Workspace ⓘ

New

Add new workspace name

Existing

abro_ado



[Why can't I see all my workspaces?](#)

I understand items in [abro_ado](#) may be deleted and can't be restored.

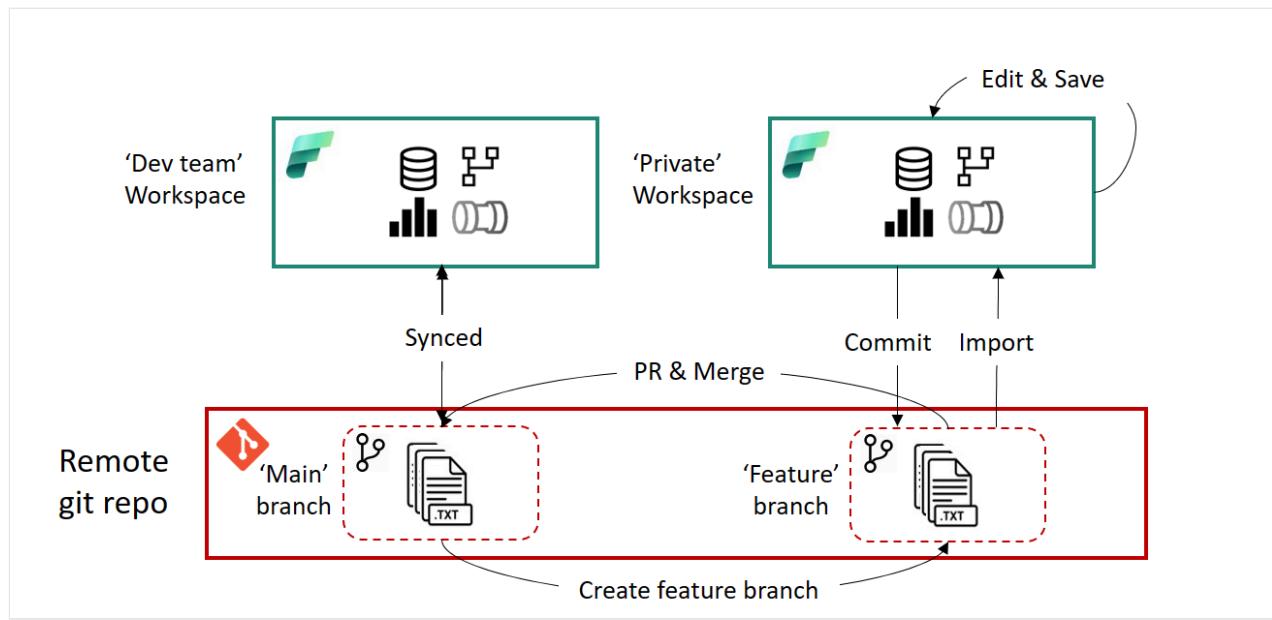
[Branch out](#)

[Cancel](#)

3. Select Branch out.

Fabric creates the new workspace and branch. You're automatically taken to the new workspace.

The workspace syncs with your feature branch, and becomes an isolated environment to work in, as illustrated. You can now work in this new isolated environment. The sync might take a few minutes. For more information on branching out, see [troubleshooting tips](#).



4. Save your changes and [commit](#) them into the feature branch.

5. When ready, create a PR to the *main* branch. The review and merge processes are done through Azure Repos based on the configuration your team defined for that repo.

Once the review and merge are complete, a new commit is created to the *main* branch. This commit prompts the user to update the content in the Dev team's workspace with the merged changes.

For more information, see [branching out limitations](#).

Release process

The release process begins once new updates complete a Pull Request process and merge into the team's shared branch (such as *Main*, *Dev*, etc.). From this point, There are different options to build a release process in Fabric. To read about different options to consider when designing your workflow, see [release process](#).

Switch branches

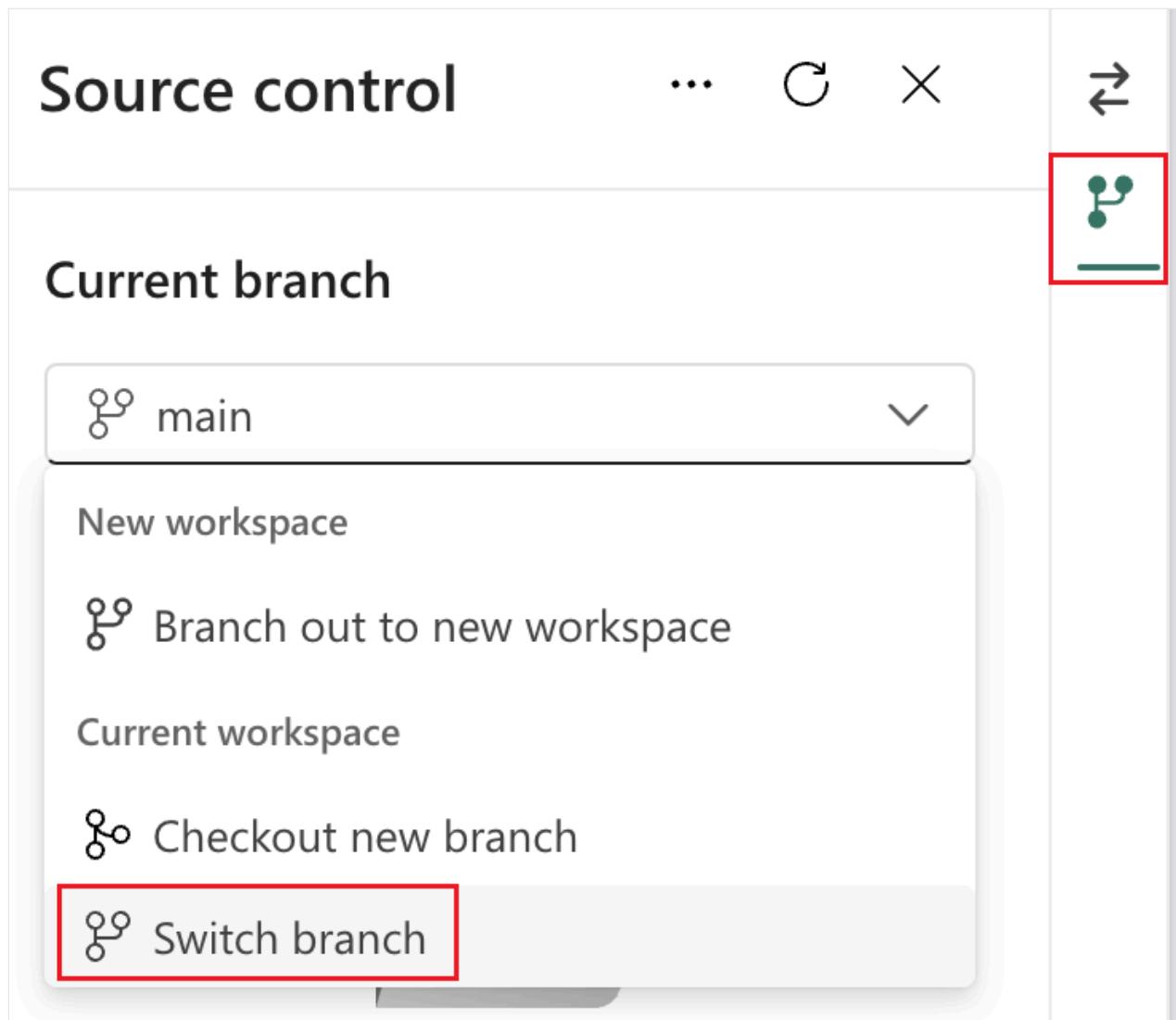
If your workspace is connected to a Git branch and you want to switch to another branch, you can do so quickly from the **Source control** pane without disconnecting and reconnecting. When you switch branches, the workspace syncs with the new branch and all items in the workspace are overridden. If there are different versions of the same item in each branch, the item is replaced. If an item is in the old branch, but not the new one, it gets deleted.

Important

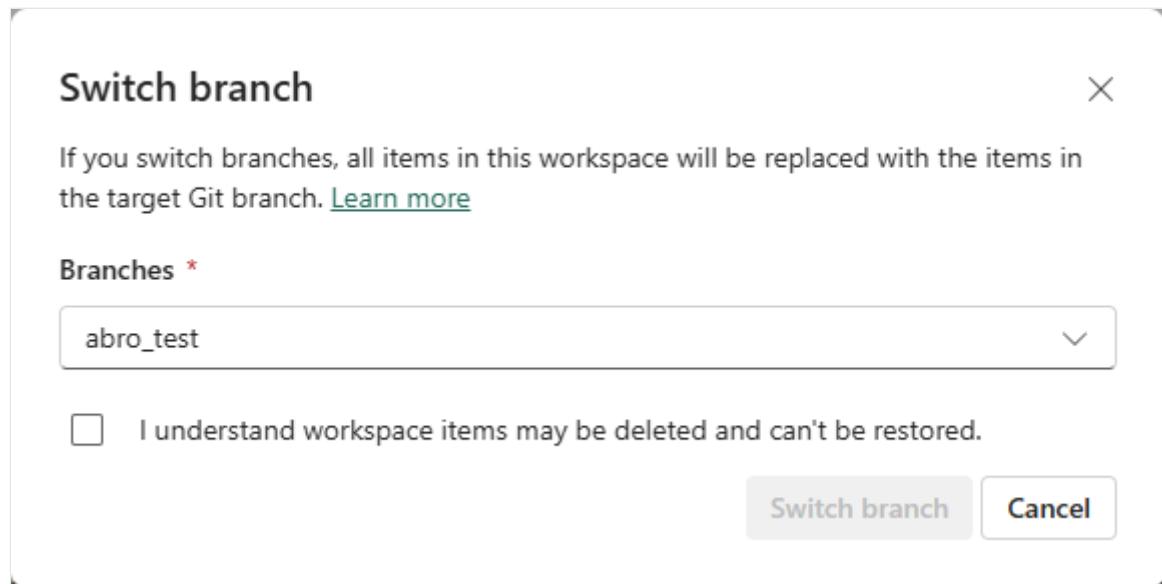
When switching branches, if the workspace contains an item in the old branch but not the new one, the item is deleted.

To switch between branches, follow these steps:

1. From the *Branches* tab of the **Source control** menu, select **Switch branch**.



2. Specify the branch you want to connect to or create a new branch. This branch must contain the same directory as the current branch.
3. Place a check in **I understand workspace items may be deleted and can't be restored.** and select **Switch branch**.



You can't switch branches if you have any uncommitted changes in the workspace. Select **Cancel** to go back and commit your changes before switching branches.

To connect the current workspace to a new branch while keeping the existing workspace status, select **Checkout new branch**. Learn more about checking out a new branch at [Resolve conflicts in Git](#).

Related content

- [Resolve errors and conflicts](#)
- [Git integration best practices](#)

Last updated on 12/15/2025

Conflict resolution

A conflict occurs when changes are made *to the same item* in both the workspace and the remote Git repository. When a conflict occurs, the Git status says **Conflict** and Commit is disabled.

	Name	Git status	Type
00	Cool Blue	✖ Conflict	Report

When you select **Update** when there are conflicts, a message notifies you that you need to resolve the conflicts before you can update.

Source control



!!!BugBashDan



× To update all to this workspace, you'll need to resolve 22 conflicts. [Learn more about resolving conflicts](#)

Changes 22 • Updates 22 •

Item	Status
All changes for Q4 incl...	×
CloudRLS v32	×
rqqmgl	×
myzmp	×
kxrfr	×
brsep	×
nhznv	×

Update all

There are three ways to resolve a conflict:

- [Select which version to keep](#) through the UI.
- [Revert](#) either the workspace or the Git repository to a previous synced state.
- [Resolve](#) the conflict in Git.
- [Manually update](#) the workspace if one or more items fail to update.

Resolve conflict in UI

Select **Update all** to see a list of all the items that have conflicts. You can then select which version to keep for each item. For each conflicted item, you can choose to accept the incoming changes from the Git repository or keep the current version that's in the workspace.

Merge and update content to resolve conflicts

Choose which version you want for the following items:

⚠ Some items may be permanently deleted and can't be restored. [Learn more](#)

Name	Accept incoming changes	Keep current content
All visual types v3	<input checked="" type="radio"/>	<input type="radio"/>
All visual types v3	<input type="radio"/>	<input checked="" type="radio"/>
Notebook_1	<input type="radio"/>	<input checked="" type="radio"/>
lakehouse1	<input checked="" type="radio"/>	<input type="radio"/>

I understand workspace items may be deleted and can't be restored.

Merge And Update **Cancel**

- Choose **Accept incoming changes** to override the changes in the workspace. The workspace changes are lost and the Git status changes to *synced* if the import succeeds.

(!) Note

Accepting incoming changes will override the current item in the workspace.

- Choose **Keep current content** to keep the version currently in the workspace. After the update is complete, the Git status becomes *uncommitted changes* as the changes in the workspace aren't yet committed to the branch.

Revert to a previous state

You can revert either the entire workspace or Git branch to last synced state. If you revert the Git branch to a previous commit, you can still see the changes made to the unsynced branch. If you revert the workspace, you lose all the changes made to the workspace since the last commit.

To revert to the prior synced state, do one of the following actions:

- Use the [Undo](#) command to return conflicted items in the workspace to their last synced state.
- Revert to the last synced state in Git using the `git revert` command in Azure DevOps.

You can also resolve conflicts by disconnecting and reconnecting the workspace. When you reconnect, [select the direction](#) you want to sync. Note, however, that when you reconnect, it overwrites all items in the workspace or branch and not just the conflicted ones. It doesn't return the workspace or branch to the last synced state. Rather, it overwrites all the content in one location with the content of the other.

Resolve conflict in git

If you're not sure what changes were made and which version to choose and don't want to revert to a previous state, you can try resolving the conflict in the Git repo by creating a new branch, resolving the conflict in that branch, and syncing it with the current one.

 **Note**

Only a workspace admin can reconnect the workspace to the new branch.

1. From the **Source control** panel, check out a new branch using the last synced branch ID shown on bottom of screen

Source control

... ⌂ X ↵



Current branch

🔗 main



New workspace

🔗 Branch out to new workspace

Current workspace

🔗 Checkout new branch

🔗 Switch branch



🔗 test56565



Last synced: 4/3/2023 at 11:06 AM

b33346e7

This step creates a new branch from the conflicted branch using the last synced Git state, before changes were made that conflict with your changes. You can see your changes in the **Source control** panel, but there's nothing to update from the Git branch. The *checkout branch* keeps the current workspace state, so uncommitted changes are retained when changing the branch.

2. Commit your changes into the new branch. This new branch now has the changes you made to the items connected to an earlier version of the Git branch that doesn't conflict with your changes.
3. In git, resolve the conflicts between the original branch and the new branch.
4. In git, merge the new branch into the original branch
5. In Fabric, **switch** the workspace back to the original branch.

Related content

- [Manually update after a failed update](#)
 - [Lifecycle management Frequently asked questions](#)
-

Last updated on 12/15/2025

Manual update

When you update items in a workspace or *undo* a commit, there's always a chance that one or more items will fail. The workspace fails to update if the incoming update from the Git branch causes inconsistencies or other problems in the workspace that are difficult to determine in advance. When an item fails to update, the update process stops. If the item that failed is the first (or only) item that was updating, the workspace remains synced with the original branch. If an item fails after one or more items succeeded in updating, you have a situation where some items are synced and some aren't. In this case, your workspace isn't synced to any Git branch.

An update can fail for many reasons. Possible reasons include, but aren't limited to, the following problems:

- Dependent items were changed or deleted
- Circular dependencies were created
- Item was renamed
- Invalid item data

ⓘ Note

This is not the same as [conflict resolution](#). If changes were made to the same item in both the workspace and Git branch, it causes a conflict and updates are disabled. This article discusses what to do a workspace fails to update even though there are no direct conflicts.

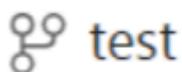
An error message lets you know which item failed and why.



This error message appears when the update fails. Any user trying to access the workspace after that sees the following error until the update is completed successfully.

Source control

X



test



Unable to update from Git

[View details](#)

Changes

Updates 1 •



Item

Status



FamilyBM2M v3



Select [View details](#) to see the previous error message.

Git statuses after an item fails

Your workspace now contains the following items:

- The item or items updated before the failure. These items are identical to the items in git, but the metadata isn't updated. They have a Git status of *synced* but with a triangle warning sign.⚠
- The item that failed. This item has a Git status of *Update required*.
- Possibly, items that weren't updated yet when the item failed. These items haven't been updated yet and have a Git status of *Update required*.

The status bar at the bottom of the screen that shows the latest sync status is red and indicated the partially synced status.



dev/Test



Partially synced: 4/20/2023 at 3:00 PM

Update the failed item

To update the workspace manually after it failed to update automatically:

1. Figure out which item is causing the update to fail and what the problem is using the error dialog that says which item failed and error message.
2. Fix the problem in Git. This can mean doing one or more of the following depending on what the issue is:
 - Revert the item to an earlier version that doesn't fail
 - Edit the item resolve the problem
 - Restore a dependant item that was deleted
 - Delete unsupported item that depends on the deleted item
3. Go back to the workspace and [Update](#) the entire workspace again.

Considerations and limitations

The update process fails as soon as one item fails. Therefore, there's no way to know if other items in the Git branch are also problematic. If you're updating many items and more than one item is problematic, you have to repeat this process once for each failed item.

Related content

[Conflict resolution](#)

Last updated on 12/15/2025

Resolve dependency errors

This article explains what dependency errors are and how to resolve them.

What is a dependency?

If you connect a workspace containing unsupported items to an empty git branch, the unsupported items aren't copied to the git branch and can't be accessed by it. You can perform any actions you want on the supported items, but the unsupported items are unseen by git.

For example, here's a sample workspace connected to a Git repository. The workspace contains a *.pbix* file, report, and semantic model. The report is dependent on the semantic model because the report refers to data from the semantic model to render. The *.pbix* file refers to both the report and the semantic model and is therefore dependent on both of them. Reports and semantic models are both supported items, while *.pbix* files aren't supported.

Name	Git status	Type
Europe	Update Required	Report
Europe	Update Required	Dataset
Europe.pbix	Unsupported	Dashboard

If you try to delete an item from a workspace, and a different, unsupported item in that workspace is dependent on it, you can't delete it.

For example, if you delete the semantic model in the previous example, it would break the dependency on the *.pbix* file and the report. If you then try to switch branches or update, you get a message that the action can't be completed.



You can delete the report from git, but you can't delete the `.pbix` file because unsupported items aren't in the git branch.

Try to switch to branch with unsupported item

Solution:

1. Use the [lineage view](#) to help you figure out which unsupported item has the dependency (in the previous example, it's the `.pbix` file).
2. Manually remove the dependency. The easiest way to do this is to delete the item.
3. Switch branches or update again.

Related content

[Maintain your git branches](#)

Last updated on 12/15/2025

Git integration license change

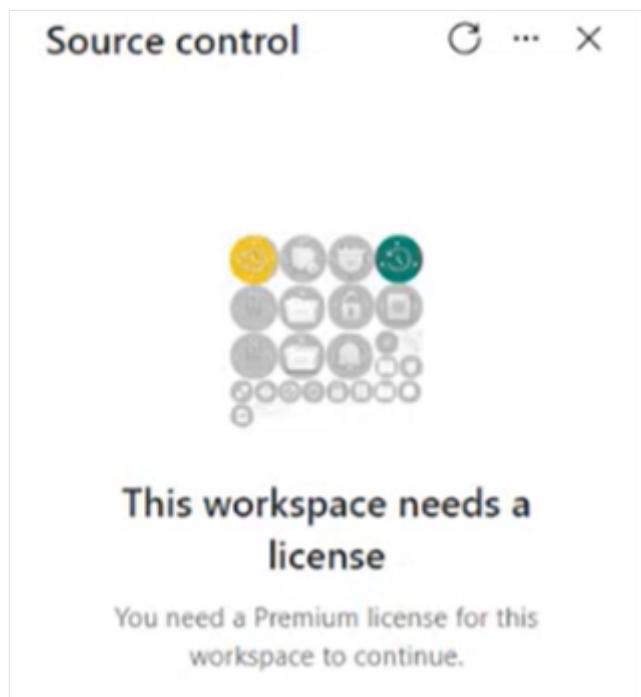
You can only connect to Git repos if you have a valid Premium license. If your license expires or if you change your license to a license that doesn't include Git integration, you can no longer connect to Git repos. This applies to trial licenses as well.

What happens when your license expires

If your workspace is connected to a Git repo, and then your license expires or you change to a different license that doesn't include Git integration, the Git-integration feature stops working and you see the following changes in your workspace homepage:

Source control view

The source control view opens automatically and shows the following error:



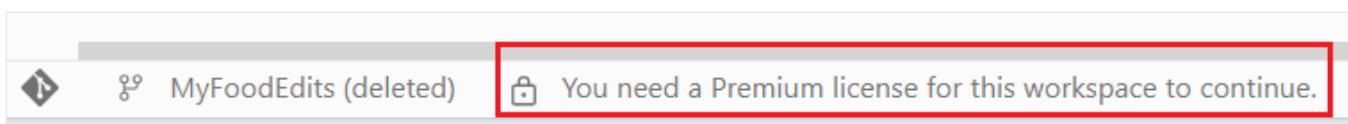
Git status

The [Git status](#) column is blank and no longer displays the item's status.

Name	Git status	Type
report2	—	Dataset
Table111	—	Dataset

Sync information

At the bottom of the workspace, instead of the [sync information](#), you see the following message:



Remove the Git connection

Without a valid Premium license, none of the Git integration features work. Unless you renew or upgrade your license, all you can do is [disconnect](#). To disconnect, go to the [Git integration](#) settings and select **Disconnect**.

Your workspace returns to a disconnected state and you can continue working in the workspace without Git.

Related content

[Manage Git branches](#)

Last updated on 12/15/2025

Git integration source code format

Items in Microsoft Fabric are stored in a folder. The folder containing the item can either be in the root directory or a subdirectory. When you connect your workspace to git, connect to the folder containing the items. Each item in the folder is represented in its own subdirectory.

Directory name

When you save a new item in Git, Git integration automatically creates a directory for that item.

The item directory name is based on the following rules:

- The pattern for the name is `{display name}.{public facing type}`.
- If necessary, the following changes to the display name are made:
 - Invalid characters are replaced with the [HTML number](#).
 - Leading space is replaced with its [HTML number](#).
 - Trailing space or dot is replaced with its [HTML number](#).
- If that folder name isn't available, the name of the item's logicalID (GUID) is used instead of the display name.

For example, if you have the following items in a workspace (note that the first and third items have an invisible leading and trailing space respectively):

<input type="checkbox"/>		Name
		ParamsUsage - V3
		ParamsUsage - V3
		ParamsUsage - V3
		ParamsUsage -*V3
		ParamsUsage*-:V3
		ParamsUsage:-*V3 .

The following directories are created in the Git repository:

Git-integration-test

- >  ParamsUsage - V3.Report
- >  ParamsUsage - V3.Report
- >  ParamsUsage - V3 .Report
- >  ParamsUsage -*V3.Report
- >  ParamsUsage*-:V3 .Report
- >  ParamsUsage:-*V3 ..Report

- Once created, Git integration never changes the name of a directory. Even if you change the name of the item, the directory name stays the same.
- If you manually change the name of an item directory, make sure to take the item's dependencies into account. For example, if you change a semantic model's directory then you should make sure to update the path of the semantic model in the report's dependency file. Keep in mind that dependency locations vary between different Fabric experiences. Changing the directory name *doesn't* cause an incoming change in the workspace.

Directory content

Each item directory contains the [item definition files](#) and [automatically generated system files](#).

Item definition files

Each item's directory has specific, required files that define that item.

The following items are currently supported in Microsoft Fabric:

- [Mirrored databases](#)
- [Notebook](#)
- [Paginated report](#)
- [Report](#)
- [Semantic model](#)

Mirrored databases

Mirrored database folders contain a `.json` file defining the mirrored database.

For instructions on using Git integration with mirrored databases, see [CI/CD for mirrored databases](#).

Notebook files

Notebook folders contain a `.py` file:

For instructions on using Git integration with notebooks, see [Notebook source control and deployment](#).

Paginated report files

Paginated report folders contain an `.rdl` file defining the paginated report. RDL (Report Definition Language) is an XML representation of a paginated report definition.

For more information about RDL, see [Report Definition Language \(RDL\)](#). For instructions on using Git integration with paginated reports, see [Git integration with paginated reports](#).

Report files

Report folders contain the following files:

- `definition.pbir`
- `report.json`

For more information about report folders and a complete list of their contents, see [Power BI Desktop project report folder](#).

Semantic model files

Semantic model folders contain the following files:

- `definition.pbism`
- `\definition` folder with TMDL files

For more information about semantic model folders and a complete list of their contents, see [Power BI Desktop project semantic model folder](#).

Automatically generated system files

In addition to the item definition files, each item directory contains one or two automatically generated system files, depending on which version you're using:

- A version 1 directory contains `item.metadata.json` and `item.config.json`. With V1, both files must be in the directory.
- A version 2 directory contains `.platform`. This file includes the content of both `item.metadata.json` and `item.config.json` files. If you have this file, you can't have the other two files. If you're using version 1 and you commit changes, your system files are automatically updated to this version.

 **Note**

Your directory must contain either the `item.metadata.json` and `item.config.json` files or the `.platform` file. You can't have all three files.

Version 2

Platform file

In version 2, instead of having two source files in each item directory, the `.platform` file combines all the information into one file along with a `$schema` property. If you have this file, you can't have the other two files.

JSON

```
{
  "version": "2.0",
  "$schema": "https://developer.microsoft.com/json-
schemas/fabric/platform/platformProperties.json",
  "config": {
    "logicalId": "e553e3b0-0260-4141-a42a-70a24872f88d"
  },
  "metadata": {
    "type": "Report",
    "displayName": "All visual types",
    "description": "This is a report"
  }
}
```

The `.platform` file contains the following attributes:

- `version`: Version number of the system files. This number is used to enable backwards compatibility. Version number of the item might be different.
- `logicalId`: (string) An automatically generated cross-workspace identifier representing an item and its source control representation.
- `type`: (string) The item's type (semantic model, report etc.)

- `displayName`: (string) The name of the item.
- `description`: (optional string) Description of the item.

If you rename the artifact in the workspace and the artifact folder in Git has a '.' suffix, then after committing, the `displayName` and directory name in Git will match.

The `logicalId` connects an item in a workspace with its corresponding item in a Git branch. Items with the same `logicalIds` are assumed to be the same. The `logicalId` preserves the link even if the name or directory change. Since a branch can be synced to multiple workspaces, it's possible to have items in different workspaces with the same `logicalId`, but a single workspace can't have two items with the same `logicalId`. The `logicalId` is created when the workspace is connected to a Git branch or a new item is synced. The `logicalId` is necessary for Git integration to function properly. Therefore, it's essential not to change it in any way.

!*Note*

When you commit changes to Git in version 1, the system files are automatically updated to version 2 along with the changes. Also, any new files exported from Power BI Desktop developer mode will be saved in the version 2 file format.

!*Note*

- The `type` field is case-sensitive. Don't change the way it's automatically generated or it might fail.
- Though you should not generally change the `logicalId` or `display name` of an item, one exception might be if you're creating a new item by copying an existing item directory. In that case, you do need to change the `logicalId` and the `display name` to something unique in the repository.

Related content

[Get started with Git integration.](#)

Automate Git integration by using APIs

The Microsoft Fabric [Git integration](#) tool enables teams to work together using source control to build an efficient and reusable release process for their Fabric content.

With [Microsoft Fabric REST APIs](#), you can automate Fabric procedures and processes to complete tasks faster and with fewer errors. This efficiency leads to cost savings and improved productivity.

This article describes how to use the [Git integration REST APIs](#) to automate Git integration in Microsoft Fabric.

Prerequisites

To work with Fabric Git APIs, you need:

- The same [prerequisites you need to use Git integration in the UI](#).
- A Microsoft Entra token for Fabric service. Use that token in the authorization header of the API call. For information about how to get a token, see [Fabric API quickstart](#).
- If you're using a service principal, it needs the same permissions as a user principal. To set up a service principal for Azure DevOps, see [Git integration with service principal](#).

You can use the REST APIs without [PowerShell](#), but the scripts in this article use PowerShell. To run the scripts, take the following steps:

- Install [PowerShell](#).
- Install the [Azure PowerShell Az module](#).

Git integration API functions

The [Git integration REST APIs](#) can help you achieve the continuous integration and continuous delivery (CI/CD) of your content. Here are a few examples of what can be done by using the APIs:

- [Connect](#) and [disconnect](#) a specific workspace from the Git repository and branch connected to it. ([Connect](#) requires the [connectionId of the Git provider credentials](#).)
- [Get connection](#) details for the specified workspace.
- [Get or create Git provider credentials connection](#).

- [Update my Git credentials](#) to update your Git credentials configuration details. Requires the [connectionId](#) of the Git provider credentials.
- [Get my Git credentials](#) to get your Git credentials configuration details.
- [Initialize a connection](#) for a workspace that is connected to Git.
- See which items have incoming changes and which items have changes that weren't yet committed to Git with the [Git status](#) API.
- [Commit](#) the changes made in the workspace to the connected remote branch.
- [Update the workspace](#) with commits pushed to the connected branch.

Examples

Use the following PowerShell scripts to understand how to perform several common automation processes. To view or copy the text in a PowerShell sample, use the links in this section. You can also see all the examples in the [Fabric Git integration samples](#) GitHub repo.

Connect and update

This section describes the steps involved in connecting and updating a workspace with Git.

For the complete script, see [Connect and update from Git](#). (The script compatibility is PowerShell 5.1)

1. **Connect to Azure account and get access token** - Sign in to Fabric as a user or a service principal. Use the [Connect-AzAccount](#) command to connect. To get an access token, use the [Get-AzAccessToken](#) command, and [convert the secure string token to plain text](#)

Your code should look something like this:

User principal

PowerShell

```
$global:resourceUrl = "https://api.fabric.microsoft.com"  
  
$global:fabricHeaders = @{  
  
function SetFabricHeaders() {  
  
    #Login to Azure  
    Connect-AzAccount | Out-Null
```

```

# Get authentication
$secureFabricToken = (Get-AzAccessToken -AsSecureString -ResourceUrl
$global:resourceUrl).Token

# Convert secure string to plain text
$ssPtr =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($secureFabricT
oken)
try {
    $fabricToken =
[System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($ssPtr)
} finally {
    [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($ssPtr)
}

$global:fabricHeaders = @{
    'Content-Type' = "application/json"
    'Authorization' = "Bearer {0}" -f $fabricToken
}
}

```

2. Call the [Connect](#) API to connect the workspace to a Git repository and branch. (you might need to [create a connection](#) first)

For information on how to obtain the Connection details (ID, Name), refer to [Get](#) or [create Git provider credentials connection](#).

Azure DevOps

PowerShell

```

$global:baseUrl = "https://api.fabric.microsoft.com/v1"
$workspaceName = "<WORKSPACE NAME>"
$getWorkspacesUrl = "{0}/workspaces" -f $global:baseUrl
$workspaces = (Invoke-RestMethod -Headers $global:fabricHeaders -Uri
$getWorkspacesUrl -Method GET).value

# Find the workspace by display name
$workspace = $workspaces | Where-Object {$_.DisplayName -eq $workspaceName}

# Connect to Git
Write-Host "Connecting the workspace '$workspaceName' to Git."
$connectUrl = "{0}/workspaces/{1}/git/connect" -f $global:baseUrl,
$workspace.Id

# AzureDevOps details
$azureDevOpsDetails = @{
    gitProviderType = "AzureDevOps"
    organizationName = "<ORGANIZATION NAME>"
}

```

```

projectName = "<PROJECT NAME>"
repositoryName = "<REPOSITORY NAME>"
branchName = "<BRANCH NAME>"
directoryName = "<DIRECTORY NAME>"
}

$connectToGitBody = @{}
#Leave only one of the following two (delete the other one):
#-----
-----
# 1. Automatic (SSO)
$connectToGitBody = @{
    gitProviderDetails = $gitProviderDetails
} | ConvertTo-Json
#-----
-----
# 2. ConfiguredConnection (User or service principal)
# Get workspaces
$connectionName = "<CONNECTION Name>"
$getConnectionsUrl = "{0}/connections" -f $global:baseUrl
$connections = (Invoke-RestMethod -Headers $global:fabricHeaders -Uri
$getConnectionsUrl -Method GET).value

# Find the connection by display name
$connection = $connections | Where-Object {$_.DisplayName -eq
$connectionName}
$connectToGitBody = @{
    gitProviderDetails = $azureDevOpsDetails
    myGitCredentials = @{
        source = "ConfiguredConnection"
        connectionId = $connection.id
    }
} | ConvertTo-Json
#-----
-----

Invoke-RestMethod -Headers $global:fabricHeaders -Uri $connectUrl -Method
POST -Body $connectToGitBody

```

3. Call the [Initialize Connection](#) API to initialize the connection between the workspace and the Git repository/branch.

PowerShell

```

# Initialize Connection

Write-Host "Initializing Git connection for workspace '$workspaceName'."

$initializeConnectionUrl = "{0}/workspaces/{1}/git/initializeConnection" -f
$global:baseUrl, $workspace.Id
$initializeConnectionResponse = Invoke-RestMethod -Headers
$global:fabricHeaders -Uri $initializeConnectionUrl -Method POST -Body "{}"

```

4. Based on the response from the Initialize Connection API, call either the [Update From Git](#) API to complete the update, or do nothing if no action required.

The following script updates and [monitors the progress](#):

PowerShell

```
if ($initializeConnectionResponse.RequiredAction -eq "UpdateFromGit") {  
  
    # Update from Git  
    Write-Host "Updating the workspace '$workspaceName' from Git."  
  
    $updateFromGitUrl = "{0}/workspaces/{1}/git/updateFromGit" -f  
    $global:baseUrl, $workspace.Id  
  
    $updateFromGitBody = @{  
        remoteCommitHash = $initializeConnectionResponse.RemoteCommitHash  
        workspaceHead = $initializeConnectionResponse.WorkspaceHead  
    } | ConvertTo-Json  
  
    $updateFromGitResponse = Invoke-WebRequest -Headers $global:fabricHeaders -  
    Uri $updateFromGitUrl -Method POST -Body $updateFromGitBody  
  
    $operationId = $updateFromGitResponse.Headers['x-ms-operation-id']  
    $retryAfter = $updateFromGitResponse.Headers['Retry-After']  
    Write-Host "Long Running Operation ID: '$operationId' has been scheduled  
    for updating the workspace '$workspaceName' from Git with a retry-after time of  
    '$retryAfter' seconds." -ForegroundColor Green  
  
    # Poll Long Running Operation  
    $getOperationState = "{0}/operations/{1}" -f $global:baseUrl, $operationId  
    do  
    {  
        $operationState = Invoke-RestMethod -Headers $global:fabricHeaders -Uri  
        $getOperationState -Method GET  
  
        Write-Host "Update from Git operation status:  
        $($operationState.Status)"  
  
        if ($operationState.Status -in @("NotStarted", "Running")) {  
            Start-Sleep -Seconds $retryAfter  
        }  
    } while($operationState.Status -in @("NotStarted", "Running"))  
}
```

Update from Git

In this section, we describe the steps involved in updating a workspace with the changes from Git. In this script, we update the workspace items with changes from Git, but we leave the Git repository unchanged.

For the complete script, see [Update workspace from Git](#).

1. Log into Git and get authentication.
2. Call the [Get Status](#) API to build the update from Git request body.
3. Call the [Update From Git](#) API to update the workspace with commits pushed to the connected branch.

Commit all

This section gives a step by step description of how to programmatically commit all changes from the workspace to Git.

For the complete script, see [Commit all changes to Git](#).

1. Log into Git and get authentication.
2. Connect to workspace.
3. Call the [Commit to Git](#) REST API.
4. Get the Long Running OperationId for polling the status of the operation.

Selective Commit

This section describes the steps involved in committing only specific changes from the workspace to Git.

For the complete script, see [Commit select changes to Git](#).

1. Log into Git and get authentication.
2. Connect to workspace.
3. Call the [Get status](#) API to see which items workspace were changed.
4. Select the specific items to commit.
5. Call the [Commit to Git](#) API to commit the selected changes from the workspace to the connected remote branch.

Monitor the progress of long running operations

For the complete script, see [Poll a long running operation](#).

1. Retrieve the operationId from the [Update From Git](#) or the [Commit to Git](#) script.
2. Call the [Get LRO Status](#) API at specified intervals (in seconds) and print the status.

Get or create Git provider credentials connection

In order to [connect](#) to a Git repository or [update your Git credentials](#) you need to provide a *connectionId*. The *connectionId* can come from either a new connection that you create, or an existing connection.

- [Create a new connection](#) with your Git provider credentials
- [Use an existing connection](#) that you have permissions for.

Create a new connection that stores your Git credentials

Azure DevOps

The following code snippet shows a sample request body to create a connection that stores your Azure DevOps credentials. The full example can be found in the [Fabric samples repo ↗](#).

PowerShell

```
# Connection with ServicePrincipal details for AzureDevOpsSourceControl
$adoSPConnection = @{
    connectivityType = "ShareableCloud"
    displayName = "<CONNECTION NAME>"
    connectionDetails = @{
        type = "AzureDevOpsSourceControl"
        creationMethod = "AzureDevOpsSourceControl.Contents"
        parameters = @(
            @{
                dataType = "Text"
                name = "url"
                value = "<Repo url in Azure DevOps>"
            }
        )
    }
    credentialDetails = @{
        credentials = @{
            credentialType = "ServicePrincipal"
            tenantId = "<SP tenant (directory) id (Guid)>"
            servicePrincipalClientId = "<SP APP (client) id (Guid)>"
            servicePrincipalSecret = "<SP Secret>"
        }
    }
}

#Note: AzureDevOps for UserPrincipal is not supported (since it requires
#interactive OAuth2)
```

Sample request

HTTP

```
POST https://api.fabric.microsoft.com/v1/connections
```

```
{  
    "displayName": "<CONNECTION NAME>",  
    "connectivityType": "ShareableCloud",  
    "connectionDetails": {  
        "creationMethod": "AzureDevOpsSourceControl.Contents",  
        "type": "AzureDevOpsSourceControl",  
        "parameters": [  
            {  
                "dataType": "Text",  
                "name": "url",  
                "value": "<Repo url in Azure DevOps>"  
            }  
        ]  
    },  
    "credentialDetails": {  
        "credentials": {  
            "credentialType": "ServicePrincipal",  
            "tenantId": "<SP tenant (directory) id (Guid)>",  
            "servicePrincipalClientId": "<SP APP (client) id (Guid)>",  
            "servicePrincipalSecret": "<SP Secret>"  
        }  
    }  
}
```

Sample response:

JSON

```
{  
    "allowConnectionUsageInGateway": false,  
    "id": "*****_****_****_****-c13b543982ac",  
    "displayName": "<CONNECTION NAME>",  
    "connectivityType": "ShareableCloud",  
    "connectionDetails": {  
        "path": "<Repo url in Azure DevOps>",  
        "type": "AzureDevOpsSourceControl"  
    },  
    "privacyLevel": "Organizational",  
    "credentialDetails": {  
        "credentialType": "ServicePrincipal",  
        "singleSignOnType": "None",  
        "connectionEncryption": "NotEncrypted",  
        "skipTestConnection": false  
    }  
}
```

Get a list of existing connections

Use the [List connections API](#) to get a list of existing connections that you have permissions for, and their properties.

Sample request

HTTP

```
GET https://api.fabric.microsoft.com/v1/connections
```

Sample response

JSON

```
{
  "value": [
    {
      "id": "e3607d15-6b41-4d11-b8f4-57cdcb19ffc8",
      "displayName": "MyGitHubPAT1",
      "gatewayId": null,
      "connectivityType": "ShareableCloud",
      "connectionDetails": {
        "path": "https://github.com",
        "type": "GitHubSourceControl"
      },
      "privacyLevel": "Organizational",
      "credentialDetails": {
        "credentialType": "Key",
        "singleSignOnType": "None",
        "connectionEncryption": "NotEncrypted",
        "skipTestConnection": false
      }
    },
    {
      "id": "3aba8f7f-d1ba-42b1-bb41-980029d5a1c1",
      "displayName": "MyGitHubPAT2",
      "gatewayId": null,
      "connectivityType": "ShareableCloud",
      "connectionDetails": {
        "path": "https://github.com/OrganizationName/RepositoryName",
        "type": "GitHubSourceControl"
      },
      "privacyLevel": "Organizational",
      "credentialDetails": {
        "credentialType": "Key",
        "singleSignOnType": "None",
        "connectionEncryption": "NotEncrypted",
        "skipTestConnection": false
      }
    }
  ]
}
```

]
}

Copy the ID of the connection you want and use it in the [Git - Connect](#) or [Git - Update My Git Credentials API](#).

Considerations and limitations

- Git integration using APIs is subject to the same [limitations](#) as the Git integration user interface.
- Refreshing a semantic model using the [Enhanced refresh API](#) causes a Git *diff* after each refresh.

Related content

- [Git integration - get started](#)
- [Fabric APIs](#)
- [Git best practices](#)

Last updated on 12/15/2025

Automate git integration with a service principal in Azure DevOps

Fabric Git Integration is the foundation for organizations implementing fully automated CI/CD pipelines, enabling seamless movement of assets across Development, Test, and Production environments.

Currently, Fabric Git Integration supports two major Git providers:

- Azure DevOps
- GitHub

This article focuses on the Service Principal capability for Azure DevOps. This integration allows the Fabric user to perform git operation using a service principal.

Azure DevOps: Authentication - automatic and configured

By default, each Fabric workspace isn't connected to any Git repository. A Fabric workspace has two different ways that it can authenticate to a git repository. These processes are called:

- Automatic git credential
- Configured credential

Automatic git credential

When an admin user wants to connect a workspace to an Azure DevOps (ADO) repository, the user must first log in from the workspace settings, the system then identifies which ADO organizations the user can access within the current Fabric tenant, allowing the user to proceed with the configuration.

Once the initial connection is established, any additional user with at least contributor permissions on the same workspace doesn't need to repeat the connection process. Instead, the system attempts to authenticate the second user with the configured ADO repository. If the user lacks the necessary permissions, the Fabric Git Integration source control pane displays a red indicator.

This streamlined authentication process is known as "Automatic Git Credential".

Configured credential

With configured credential you can programmatically create an Azure DevOps cloud connection using a service principal.

The Azure DevOps connection supports two authentication methods:

- OAuth 2.0
- Service Principal

Both methods include support for [multitenant \(cross-tenant\) scenarios](#), giving organizations flexibility across environments.

Any other user with at least Contributor permissions on the same workspace doesn't need to repeat the connection process. Before service principal support, the system attempted to authenticate secondary users only through **Automatic authentication**.

If **Automatic authentication** fails, the system also attempts to connect using any **Configured Credential** which the user has access to, ensuring a smoother experience and reduces redundant setup steps.

How It Works

To connect a Fabric workspace to an external Git provider using a Service Principal, Git integration must use a Fabric cloud connection of type **Azure DevOps – Source Control**.

This cloud connection can be created in two ways through the portal:

- Manually via [Manage Connection Settings](#)
- Through **workspace settings** using the [Add Account](#) option

In both cases, the connection is created under the logged-in user's identity.

If a Service Principal needs to use this connection, the user needs to either

- share the connection with the service principal
- create a new connection using the [Connections REST API](#), passing the Service Principal credentials.

The steps below outline how to use the API to create the cloud connection using a service principal.

Prerequisites

To complete the steps outlined, you need the following permissions:

- Register an Entra ID application and note:
 - Tenant ID
 - Client ID
 - Client Secret
- Grant the Service Principal:
 - Access to the relevant Azure DevOps organization and project.
 - Admin permissions on the Fabric workspace.

Connect a new workspace to Azure DevOps using Service Principal

To connect a Fabric workspace to Azure DevOps using Service Principal programmatically, the following steps need to be followed:

1. **Generate Service Principal access token:** Authenticates with Microsoft Fabric using a service principal.
2. **Create Azure DevOps cloud connection:** Creates a new connection resource in Microsoft Fabric that stores the Azure DevOps repository credentials and configuration.
3. **Connect workspace to git:** Links a specific Fabric workspace to the Azure DevOps repository using the connection created in step 2.
4. **Initialize Connection:** Initializes the Git connection.

1. Generate Service Principal access token

The following examples show how to generate the service principal access token.

Bash

```
curl --request GET \
--url https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token \
--header 'content-type: multipart/form-data' \
--form grant_type=client_credentials \
--form client_id=<client-id> \
--form 'client_secret=<client-secret>' \
--form scope=https://api.fabric.microsoft.com/.default
```

 Note

Copy the access_token from the response for later steps.

2. Create Azure DevOps cloud connection

Creates a new connection resource in Microsoft Fabric that stores the Azure DevOps repository credentials and configuration.

Bash

```
curl --request POST \
--url https://api.fabric.microsoft.com/v1/connections \
--header 'authorization: Bearer <step1: access-token>' \
--header 'content-type: application/json' \
--data '{

"displayName": "<name of the connection>",
"connectivityType": "ShareableCloud",
"connectionDetails": {
"creationMethod": "AzureDevOpsSourceControl.Contents",
"type": "AzureDevOpsSourceControl",
"parameters": [

{
"dataType": "Text",
"name": "url",
"value": "https://dev.azure.com/<ado org name>/<project name>/_git/<repo name>/"}}],
"credentialDetails": {
"credentials": {
"credentialType": "ServicePrincipal",
"tenantId": "<tenant-id>",
"servicePrincipalClientId": "<client-id>",
"servicePrincipalSecret": "<client-secret>"}}}'
```

! Note

Save the ID from the response. It is used in the next steps.

3. Connect workspace to git

Links a specific Fabric workspace to the Azure DevOps repository using the connection created in step 2.

Bash

```
curl --request POST \
--url https://api.fabric.microsoft.com/v1/workspaces/<workspace-id>/git/connect \
--header 'authorization: Bearer <step1: access-token>' \
--header 'content-type: application/json' \
--data '{
"gitProviderDetails": {
```

```
"organizationName": "<ado org name>",
"projectName": "<project name>",
"gitProviderType": "AzureDevOps",
"repositoryName": "<repo name>",
"branchName": "<branch name>",
"directoryName": "<folder name - must exist before OR empty>"
},
"myGitCredentials": {
"source": "ConfiguredConnection",
"connectionId": "<step 2 - the new connection id>"}'
```

4. Initialize Connection

Initialize Connection, read more [here](#).

! Note

Replace < > with your values, pay attention for initializationStrategy parameter, In case the connected workspace has items already, you might consider using "preferWorkspace".

Bash

```
curl --request POST \
--url https://api.fabric.microsoft.com/v1/workspaces/<workspace-
id>/git/initializeConnection \
--header 'authorization: Bearer <step1: access-token>' \
--header 'content-type: application/json' \
--data '{"initializationStrategy": "PreferRemote"}'
```

If the repository /workspace is not empty, the response return requiredAction param (which is based on your initializationStrategy), use [update-from-git](#) OR [commit-to-git](#) accordingly with workspaceHead and remoteCommitHash from the response to finalize the process

Connect an Existing Workspace to Use Service Principal

If your workspace is already connected to Azure DevOps using a user identity, but you want to perform [Fabric Git REST API](#) operations with a Service Principal, follow these steps:

1. Add Service Principal as Workspace Admin.
2. Grant Service Principal access to the Azure DevOps Cloud Connection. You have two options:

- **Share an existing connection:** Log in with a user who has access to the relevant ADO cloud connection and share it with the Service Principal via Manage Users.
- **Create a new connection:** Repeat Steps 1 & 2 from the previous section to create a new cloud connection using Service Principal credentials.

3. Verify access - Call the GET Connections API to confirm the Service Principal can access the required cloud connection [here](#):

Bash

```
curl --request GET \  
--url https://api.fabric.microsoft.com/v1/connections \  
--header 'authorization: Bearer <step 2: access-token>'
```

Grab the **id** value of the relevant connection from the response.

4. Update Git Credential: Generate an **access token** (step 1 from previous section) and call the [Update My Git Credential API](#), read more [here](#) (Replace < > with your values):

Bash

```
curl --request PATCH \  
--url https://api.fabric.microsoft.com/v1/workspaces/<workspace-  
id>/git/myGitCredentials \  
--header 'authorization: Bearer <step 2: access-token>' \  
--header 'content-type: application/json' \  
--data '{  
"source": "ConfiguredConnection",  
"connectionId": "<step 3: connection id>"}
```

After these steps, the Service Principal is fully configured and ready to execute Fabric Git REST API operations.

Related content

- [Automate Git integration by using APIs](#)
- [Manual git integration with a service principal in Azure DevOps](#)
- [Understand the Git integration process](#)
- [Manage Git branches](#)
- [Git integration best practices](#)

Manual git integration with a service principal in Azure DevOps

This article provides manual steps on how to set up a service principal for integrating Microsoft Fabric with Azure DevOps using user identity. To learn how Azure DevOps service principal integration is working, check out our document - [Automate git integration with a service principal in Azure DevOps](#).

Prerequisites

To register an application with your Microsoft Entra tenant and use it to integrate your Fabric workspace with Git, you need to have:

- At least [Cloud Application Administrator](#) permissions.
- A Fabric workspace with Admin permissions.

Step 1: Register an application with Microsoft Entra ID

Register your application with Microsoft Entra ID, and create a secret by following the directions in [Register your app](#). Confirm that your organization's policies allow the creation of client secrets and their use for token acquisition. Be sure to save the secret, it is required in a later step.

If your application resides in a tenant that isn't the same as the home for your Azure DevOps instance, see [Multitenant considerations](#).

 Note

Be sure to save the secret. It's used in the later steps.

For more information, see [Application and service principal objects in Microsoft Entra ID](#) and [Security best practices for application properties in Microsoft Entra ID](#).

For an example of application registration and service principal creation, see [Register a Microsoft Entra app and create a service principal](#).

Step 2: Assign service principal to a DevOps organization

After creating our application and the service principal, we need to add it to our Azure DevOps organization to grant access to resources.

1. Log in to your Azure DevOps organization
2. Browse to **Organization settings -> User -> Add users**
3. Select to add the service principal

The screenshot shows the 'Add new users' dialog box overlaid on the Azure DevOps 'Users' page. The dialog box has fields for 'Users or Service Principals' (with a placeholder 'Users or Service Principals'), 'Access level' (set to 'Basic'), 'Add to projects' (empty), and a checked 'Send email invites (to Users only)' checkbox. The background shows a list of users with one entry: 'Total 1' and 'Name' (with a downward arrow). The left sidebar shows 'Organization Settings' with 'Users' selected.

4. Navigate to relevant Azure DevOps **project settings -> Teams**
5. Add the service principal to relevant team

Step 3: Create Azure DevOps source control connection

Next, we create the Azure DevOps source control connection. The following information is required to complete this step.

Obtain the tenant ID

To obtain the tenant ID, use the following steps.

1. Go to the [Azure portal](#) and sign in with your credentials.
2. Navigate to Microsoft Entra ID (Azure Active Directory)
3. Under the "Overview" section, you see your "Tenant ID" listed.

The screenshot shows the Microsoft Entra ID (Azure Active Directory) Overview page for the tenant 'Contoso'. The left sidebar includes links for Overview, Preview features, Diagnose and solve problems, Manage (with sub-links for Users, Groups, External Identities, Roles and administrators, Administrative units, Delegated admin partners, Enterprise applications, and Devices), and Alerts. The main content area has tabs for Overview, Monitoring, Properties, Recommendations, and Setup guides. A search bar at the top says 'Search your tenant'. Below it, under 'Basic information', there is a table with the following data:

Name	Contoso	Users	35
Tenant ID	aaaabbbb-0000-c000-1111-d00d2222eeee	Groups	35
Primary domain	contoso.onmicrosoft.com	Applications	5
License	Microsoft Entra ID P2	Devices	0
Alerts			

For other ways to obtain the tenant ID, see [How to find your Microsoft Entra tenant ID](#).

Obtain the Service Principal ID

To obtain the Service principal ID, use the following steps.

1. Go to the [Azure portal](#) and sign in with your credentials.
2. Navigate to Microsoft Entra ID (Azure Active Directory)
3. On the left, select **App registrations**
4. Navigating to the app and select the Overview tab
5. Use the **Application (client) ID** for the Service Principal ID

^ Essentials

Display name	: adb_app_sp
Application (client) ID	: 00001111-aaaa-2222-bbbb-3333cccc4444
Object ID	: aaaaaaaaaa-0000-1111-2222-bbbbbbbbbbbb
Directory (tenant) ID	: aaaabbbb-0000-c000-1111-d00d2222eeee
Supported account types	: My organization only

Create the source control connection

To create the source control connection, use the following details and steps.

Name	Description
Display Name	The name of the source control connection. It should be unique.
Azure DevOps URL	The url to your repository in Azure DevOps.
Authentication method	The authentication method for the connection. Service Principal should be selected
Tenant ID	The ID of the tenant where Azure DevOps is located. See the Obtain the tenant ID section.
Service principal ID	The Application (client) ID from the app overview in the Azure portal. See the Obtain Service Principal ID section.
Service principal key	That's the secret obtained in step 1.

1. From a workspace, select **workspace settings**
2. Select **Git Integration**
3. Select **Azure DevOps**
4. Click on **Add Account**
5. Under **Display name**, enter a name.
6. Enter the Azure DevOps URL.
7. Under **Authentication method**, select **Service Principal**.
8. Complete the other details (Tenant ID, Service principal ID, Service principal key) using the information from above.

Add Azure DevOps account

Display name *
Create a new name

Azure DevOps URL * ⓘ
Example: https://dev.azure.com/{organization}/{project}/_git/{repository}

Authentication method * ⓘ

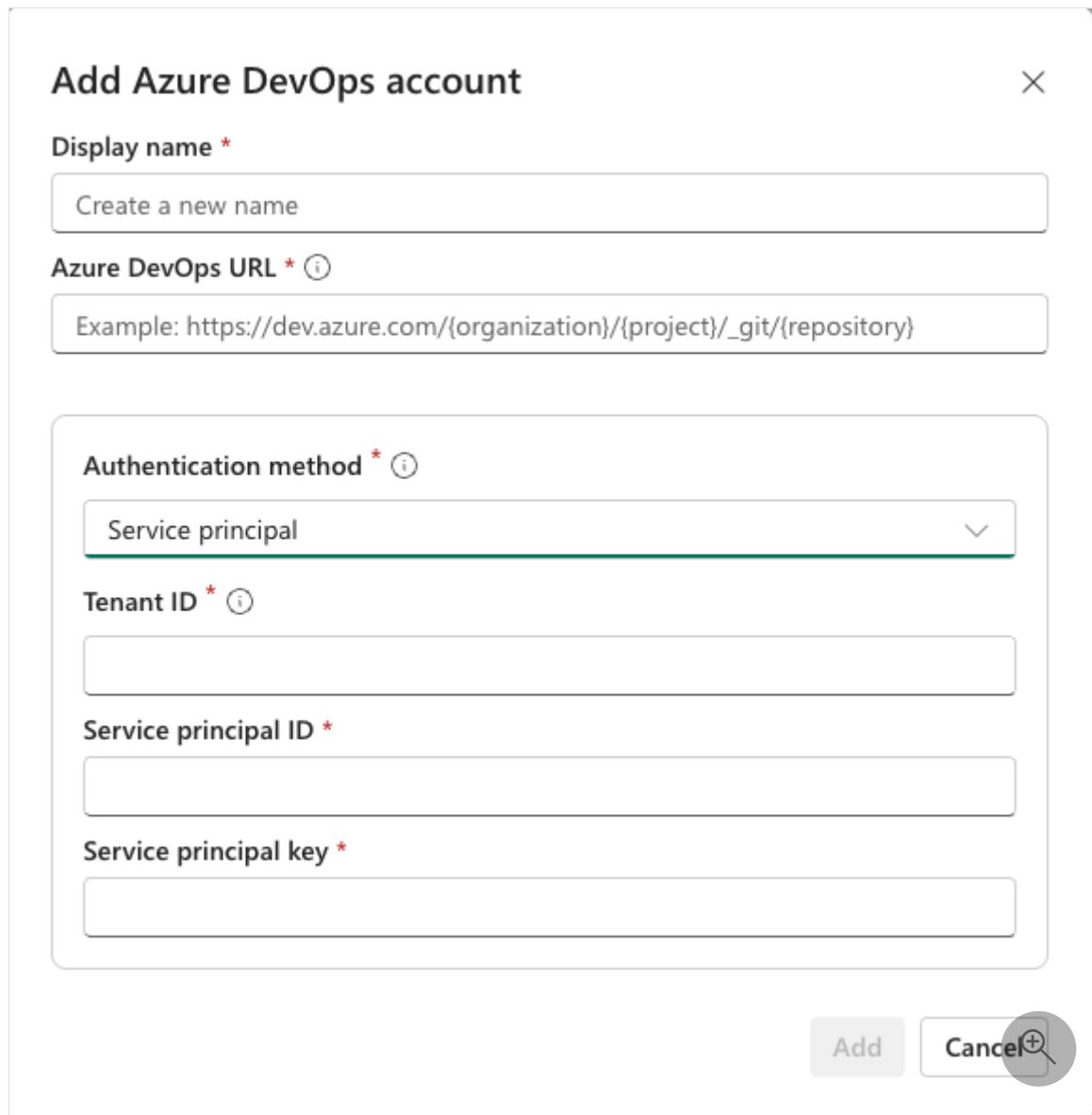
Service principal

Tenant ID * ⓘ

Service principal ID *

Service principal key *

Add Cancel

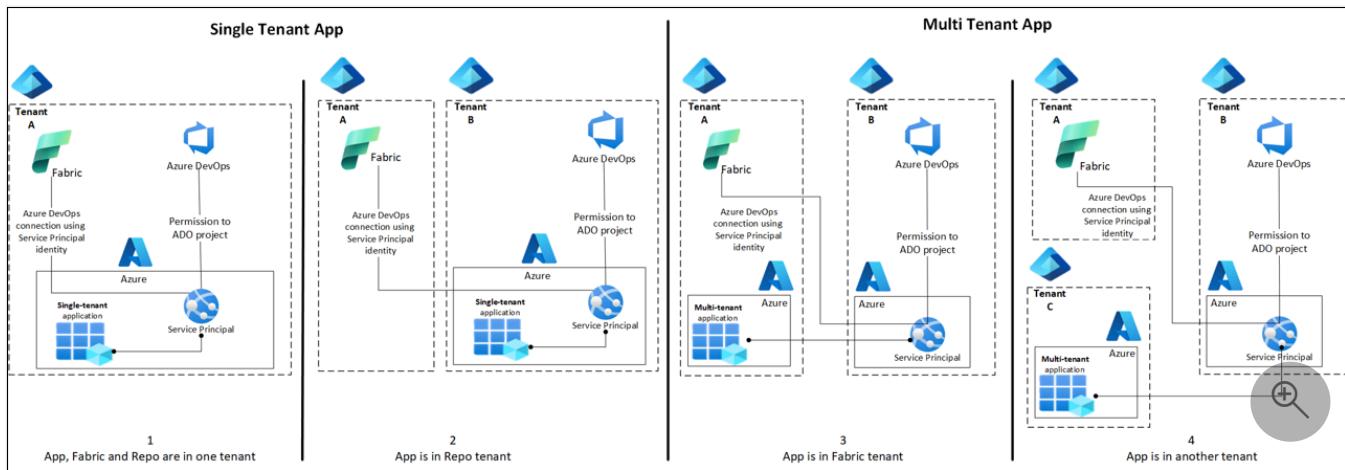


9. After adding the connection, you need to click on **connect** and complete the git connection details. For more information, see [Connect to a workspace](#)

Multitenant considerations for service principal creation

To access resources secured by a Microsoft Entra tenant, your application must have a security principal. When you create your application, the service principal is autogenerated on the tenant where the application resides.

In cases where your applications tenant is different than the home tenant of your Azure DevOps instance, you'll need to create the service principal, in the Azure DevOps tenant. Consider the following scenarios when registering your app in step 1 and see the examples.



[Expand table](#)

Scenario	Application registered as	service principal creation
1 - Fabric, DevOps, application all in same tenant	Accounts in this organizational directory only - single tenant apps	No other requirements
2 - DevOps and application in same tenant	Accounts in this organizational directory only - single tenant apps	No other requirements
3 - Fabric and application in one tenant, DevOps in a separate tenant	Accounts in any organizational directory - multitenant apps	Create SP in Azure DevOps tenant. See examples.
4 - Fabric, DevOps, and application all reside in different tenants	Accounts in any organizational directory - multitenant apps	Create SP in Azure DevOps tenant. See examples.

You can create the service principal in the Azure DevOps tenant using one of the examples. The following examples show how to do this with Azure CLI and PowerShell. Both examples assume that your application is in tenant A and Azure DevOps is in tenant B.

Azure CLI

```
az login --tenant <your-tenant-id> # where --tenant is the tenant ID of Azure DevOps tenant (tenant B)

az ad sp create --id <app id> # where --id <app id> is the client ID of the application in tenant A
```

For more information, see [How and why applications are added to Microsoft Entra ID and Tenancy in Microsoft Entra ID](#).

Appendix: Edit service principal connection details

When you need to update your service principal details, for example, update service principal key, use the following instructions:)

1. In [Fabric settings](#), navigate to **Manage Connections and Gateways**. Locate the cloud connection that you previously created.
2. Edit the connection with the updated settings.

 **Note**

If you want to create a new connection instead of editing an existing one, do this by selecting **+New** in the top left corner to add a new cloud connection.

3. Once you've finished editing the connection, click **Save**.

Related content

- [Automate Git integration by using APIs](#)
- [Automate git integration with a service principal in Azure DevOps](#).
- [Understand the Git integration process](#)
- [Manage Git branches](#)
- [Git integration best practices](#)

Last updated on 12/18/2025

Microsoft Fabric and GitHub Enterprise Cloud with data residency support (Public preview)

Microsoft Fabric is expanding its Git integration capabilities to support [GitHub Enterprise Cloud with data residency \(ghe.com\)](#) instances. These new capabilities enable enterprise customers to meet regulatory commitments while using Fabric's CI/CD workflows.

Previously, git integration supported only repositories hosted on github.com. This blocked organizations that require their GitHub Enterprise data to be stored within specific geographic boundaries. Now, Fabric workspaces are able to connect to repositories hosted on .ghe.com/<org_name>, using the same UI and API experiences already established for GitHub.com. This change treats GHE.com as part of the existing GitHub provider—not a new provider—while allowing users to authenticate via Personal Access Tokens (PATs).

How it works

Once connected, users with appropriate permissions can perform standard Git operations such as syncing, committing, updating content, and branching out from an existing workspace—even across environments where one workspace uses GitHub.com and another uses GHE.com.

All Git Integration APIs function the same across both domains, and schema updates remain fully backward-compatible.

Add GitHub account

Display name *

Personal access token * ⓘ

Repository URL ⓘ

Add Cancel +

Limitations and considerations

The following limitations apply:

- Requires a dedicated GHE.com connection per repository
- Currently, no organization-level connection support.

Related content

- [Understand the Git integration process](#)
- [Manage Git branches](#)
- [Git integration best practices](#)

Last updated on 12/16/2025

Introduction to deployment pipelines

ⓘ Note

The articles in this section describe how to deploy content to your app. For version control, see the [Git integration](#) documentation.

Microsoft Fabric's deployment pipelines tool provides content creators with a production environment where they can collaborate with others to manage the lifecycle of organizational content. Deployment pipelines enable creators to develop and test content in the service before it reaches the users. See the full list of [Supported item types](#) that you can deploy.

ⓘ Note

- The new Deployment pipeline user interface is currently in [preview](#). To turn on or use the new UI, see [Begin using the new UI](#).
- Some of the items for deployment pipelines are in preview. For more information, see the list of [supported items](#).

Learn to use deployment pipelines

You can learn how to use the deployment pipelines tool by following these links.

- [Create and manage a deployment pipeline](#) - A Learn module that walks you through the entire process of creating a deployment pipeline.
- [Get started with deployment pipelines](#) - An article that explains how to create a pipeline and perform key functions such as deployment, comparing content in different stages, and creating deployment rules.

Supported items

When you deploy content from one pipeline stage to another, the copied content can contain the following items:

- Data Engineering items:
 - [Environment](#)
 - [GraphQL](#)
 - [Lakehouse \(preview\)](#)

- [Notebook](#)
 - [Spark Job Definitions](#)
 - [User Data Functions](#)
- Data Science items:
 - [Machine learning experiments \(preview\)](#)
 - [Machine learning models \(preview\)](#)
 - [Data Agents \(preview\)](#)
- Data Factory items:
 - [Copy Job](#)
 - [Dataflows gen2](#)
 - [Pipeline](#)
 - [Mirrored database](#)
 - [Mount ADF](#)
 - [Mirrored snowflake \(preview\)](#)
- Real-time Intelligence items:
 - [Activator \(preview\)](#)
 - [Digital twin builder \(preview\)](#)
 - [Eventhouse](#)
 - [EventStream](#)
 - [KQL database](#)
 - [KQL Queryset](#)
 - [Real-time Dashboard](#)
 - [Event Schema Set \(preview\)](#)
 - [Maps \(preview\)](#)
 - [Anomaly detection \(preview\)](#)
- Data Warehouse items:
 - [Warehouse *\(preview\)](#)
 - Mirrored Azure Databricks Catalog
- Power BI items:
 - [Dashboard \(preview\)](#)
 - [Dataflow \(preview\)](#)
 - [Datamart \(preview\)](#)
 - [Org app \(preview\)](#)
 - [Paginated report \(preview\)](#)
 - [Report \(based on supported semantic models\) \(preview\)](#)
 - [Semantic model \(that originates from a .pbix file and isn't a PUSH dataset\) \(preview\)](#)
- Database items:

- [SQL database \(preview\)](#)
- [Cosmos database \(preview\)](#)
- Industry solutions:
 - [Healthcare \(preview\)](#)
 - [HealthCare Cohort \(preview\)](#)
- IQ (preview) items:
 - [Ontology \(preview\)](#)

Pipeline structure

You decide how many stages you want in your deployment pipeline. There can be anywhere from two to 10 stages. When you create a pipeline, the default three typical stages are given as a starting point, but you can add, delete, or rename the stages to suit your needs. Regardless of how many stages there are, the general concepts are the same:

- **Development**

The first stage in deployment is where you upload new content with your fellow creators. You can design build, and develop here, or in a different stage.

- **Test**

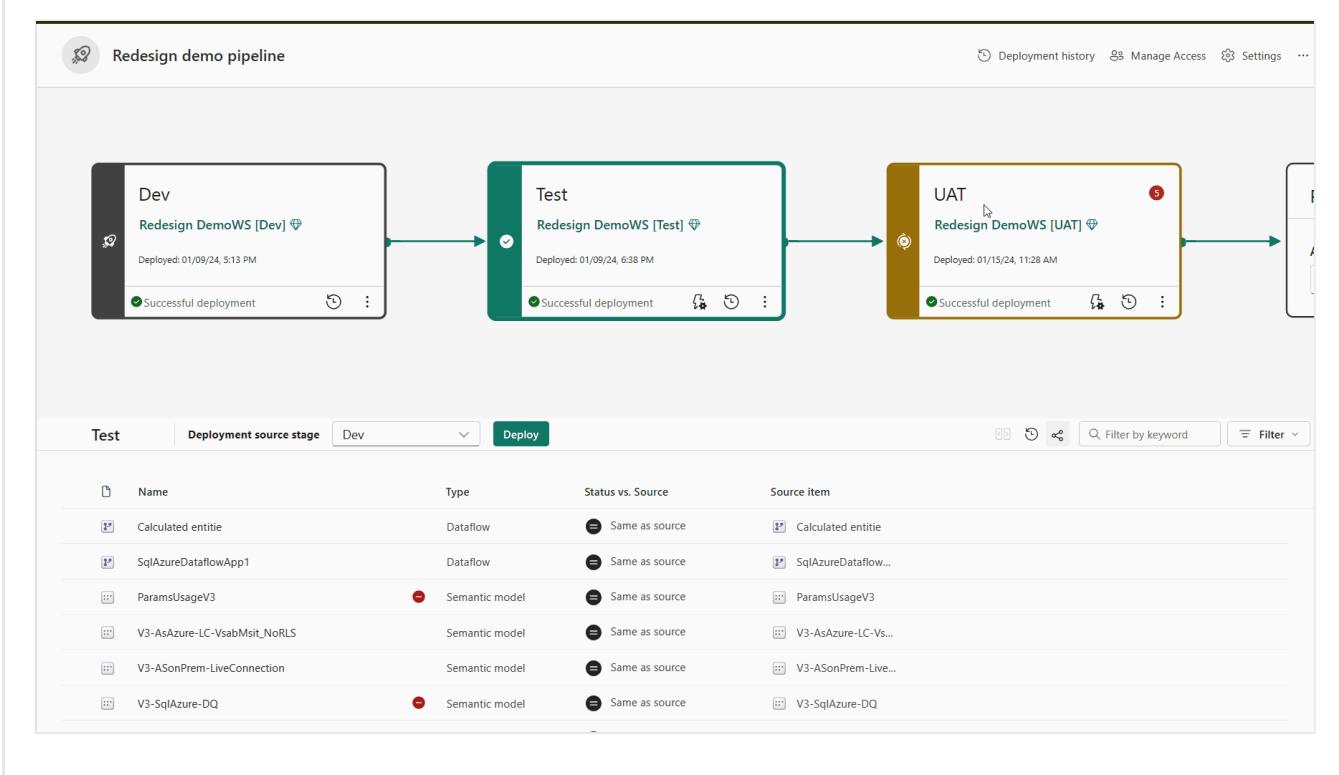
After you make all the needed changes to your content, you're ready to enter the test stage. Upload the modified content so it can be moved to this test stage. Here are three examples of what can be done in the test environment:

- Share content with testers and reviewers
- Load and run tests with larger volumes of data
- Test your app to see how it looks for your end users

- **Production**

After testing the content, use the production stage to share the final version of your content with business users across the organization.

New pipeline design



Item pairing

Pairing is the process by which an item (such as a report, dashboard, or semantic model) in one stage of the deployment pipeline is associated with the same item in the adjacent stage. Pairing occurs when you assign a workspace to a deployment stage or when you deploy new unpaired content from one stage to another (a clean deploy).

A good understanding of pairing is crucial to help you understand when items are copied, when they're overwritten, and when a deployment fails.

If items aren't paired, even if they appear to be the same (have the same name, type, and folder), they don't overwrite on a deployment. Instead, a duplicate copy is created and paired with the item in the previous stage.

Paired items appear on the same line in the pipeline content list. Items that aren't paired, appear on a line by themselves:

New pairing design

The screenshot shows a comparison table for deployment items:

Selected stage item	Type	Compared to source	Source stage item
SqlAzureDataflowApp1	Dataflow	Same as source	SqlAzureDataflowApp1
V3-SqlAzure-Cached	Report	Same as source	V3-SqlAzure-Cached
V3-SqlAzure-Cached	Semantic model	Same as source	V3-SqlAzure-Cached
V3-SqlAzure-DQ.pbix	Dashboard	Same as source	V3-SqlAzure-DQ.pbix
—	dataflow	Only in source	NewEmail2

- Items that are paired remain paired even if you change their names. Therefore, paired items can have different names.
- Items added after the workspace is assigned to a pipeline aren't automatically paired. Therefore, you can have identical items in adjacent workspaces that aren't paired.

For a detailed explanation of which items are paired and how pairing works, see [Item pairing](#).

Deployment method

Deployment pipelines provide content creators with a production environment where they can collaborate with others to manage the lifecycle of organizational content.

The deployment pipeline is composed of stages. You decide how many stages you want in your deployment pipeline. There can be anywhere from two to 10 stages. The default is 3.

The deployment process lets you clone content from one stage in the deployment pipeline to another, typically from development to test, and from test to production. During deployment, Microsoft Fabric copies the content from the source stage to the target stage.

Folders enable users to efficiently organize and manage workspace items in a familiar way.

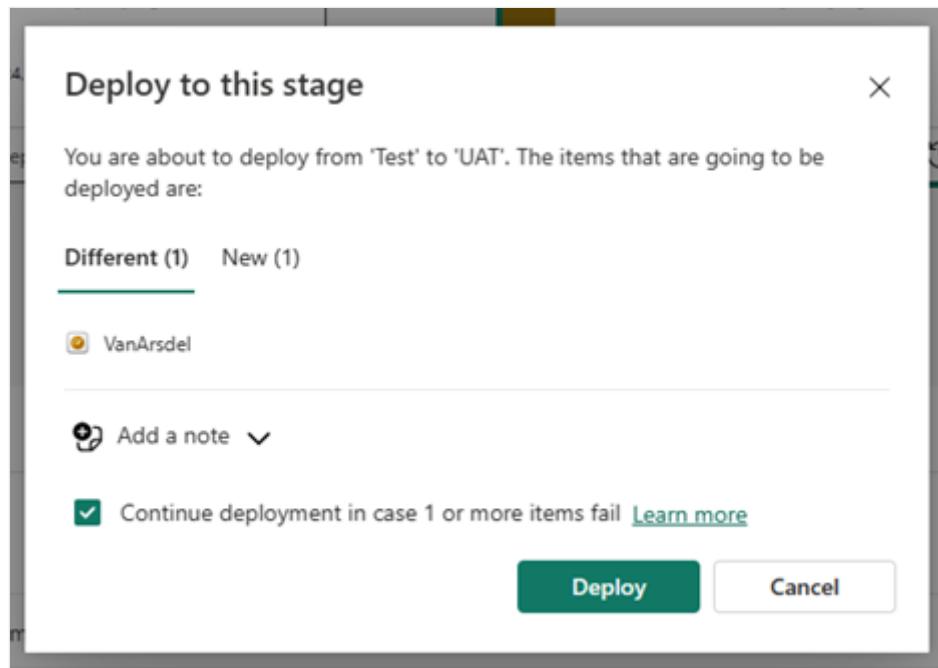
When you deploy content that contains folders to a different stage, the folder hierarchy of the applied items is automatically applied.

With the current view of the folders hierarchy, you can select for deployment, only items in the same folder level. You cannot select items across folders.

Flat list view of deployment pipelines allows you to select items regardless of its location. With the flat list view, you can select items across folders, regarding their location in the workspace. For more information, see [flat list view](#).

To deploy content to another stage, at least one item must be selected. When you deploy content from one stage to another, the items being copied from the source stage overwrite the paired item in the stage you're in according to the [pairing rules](#). Items that don't exist in the source stage remain as is.

After you select **Deploy**, you get a confirmation message.



Learn more about [which item properties are copied to the next stage](#), and which properties aren't copied, in [Understand the deployment process](#).

Automation

You can also deploy content programmatically, using the [deployment pipelines REST APIs](#). Learn more about the automation process in [Automate your deployment pipeline using APIs and DevOps](#).

Related content

- [Understand the deployment pipelines process](#)
- [Get started with deployment pipelines](#)

Last updated on 12/15/2025

New deployment pipelines user interface

The user interface for Microsoft Fabric's Deployment pipelines is undergoing a change to improve the user experience. The new UI is designed to be more focused, easier to navigate, and have a smoother flow. The functionality has stayed the same, and anything you can do with the original UI you can do with the new UI.

(!) Note

The new UI is currently in preview and is available to all users. The old UI is still available and can be accessed by using the [toggle switch](#) in the upper right corner of the page.

What changed in the new UI

The new UI offers several improvements and a change in the workflow concept over the old UI. The main changes are:

- [New workflow concept](#)
- [Enhanced experience](#)

New workflow concept

Pipeline stage perspective

The main difference in the new UI is the perspective. The focus is on a single selected stage and all operations are done and viewed from the perspective of that stage. Once you select a stage in the pipeline, you view and manage everything in the pipeline in relation to that stage. From that stage you can:

- View the content
- View the sync status
- Compare the content with the source stage
- View the deployment history of previous deployments to that stage
- Create rules for deployment to this stage

This is a more consistent and intuitive experience than before. Whereas in the original UI, you deployed content while in the source stage, but created rules while in the target stage, in the new UI, everything is done from the same stage.

Compare status by item

When you select a stage in the new UI, the content of that stage appears on the bottom pane with each item shown next to its [paired item](#) in the source stage and the sync status displayed by default. The source stage is the one shown in the drop-down menu next to the *Deploy* button. Learn more in [Compare stages content](#).

The screenshot shows a deployment pipeline interface. At the top, there are tabs for 'Test' and 'Deploy from' (set to 'Development'), a 'Deploy' button, and a 'Select related' link. Below this is a table comparing items between the source (Development) and target (Test) stages.

Selected stage item	Type	Compared to source	Source stage item
full_v3	Report	✗ Not in source	—
full_v3	Semantic model	≡ Same as source	full_v3
—	Semantic model	+ Only in source	FamilyBM2M
—	report	+ Only in source	ReportCreate1

Enhanced experience

The new UI contains advanced functionalities for a better experience. Some of these functionalities include:

- Search for an item by its name (free text)
- Filter by item type or sync status.
- Sort by name, type, or sync status
- Zoom in/out in the pipeline view
- The following functionalities are supported only in the new UI
 - [Workspace folders](#) - View the workspace items by their folder hierarchy. To deploy items in a subfolder, navigate to that folder.
 - Parent/child items - Child items are shown but can't be deployed. During deployment, the child item is recreated in the target stage in each deployment.
 - Unsupported items in the pipeline can be seen and filtered, but not deployed.
 - Custom actions for deployment pipelines, such as *Configure rules*, are no longer available in the item menu but are available elsewhere in the UI. The item menu list is now the same as the menu on the workspace page.

What remains unchanged

The following deployment pipeline features remain unchanged:

- [Deployment rules](#)
- [Deployment history](#)

- [Change review](#)
- [The Deployment pipelines home page](#)
- [Deployment pipelines entry points](#)

Begin using the new UI

The default UI is the new one, but you can switch between the new and old UIs using the toggle switch in the upper right corner.



Your selection is saved and will be remembered the next time you visit the page. You can switch back and forth as many times as you like.

Related content

- [Overview of deployment pipelines](#)
- [Get started with deployment pipelines](#)

Last updated on 12/15/2025

Get started with deployment pipelines

This article walks you through the basic settings required for using deployment pipelines in Microsoft Fabric. We recommend reading the [deployment pipelines introduction](#) and understanding [which items can be deployed](#) before you proceed.

ⓘ Note

- The new Deployment pipeline user interface is currently in [preview](#). To turn on or use the new UI, see [Begin using the new UI](#).
- Some of the items for deployment pipelines are in preview. For more information, see the list of [supported items](#).

You can also complete the [Create and manage a Fabric deployment pipeline](#) training module, which shows you step by step how to create a deployment pipeline.

Prerequisites

In a deployment pipeline, one Premium workspace is assigned to each stage. Before you start working with your pipeline in production, review the [capacity requirements](#) for the pipeline's workspaces.

To access the deployment pipelines feature, you must meet the following conditions:

- You have a [Microsoft Fabric subscription](#)
- You're an admin of a Fabric [workspace](#)

ⓘ Note

You can also see the deployment pipelines button if you previously created a pipeline or if a pipeline was shared with you.

Step 1 - Create a deployment pipeline

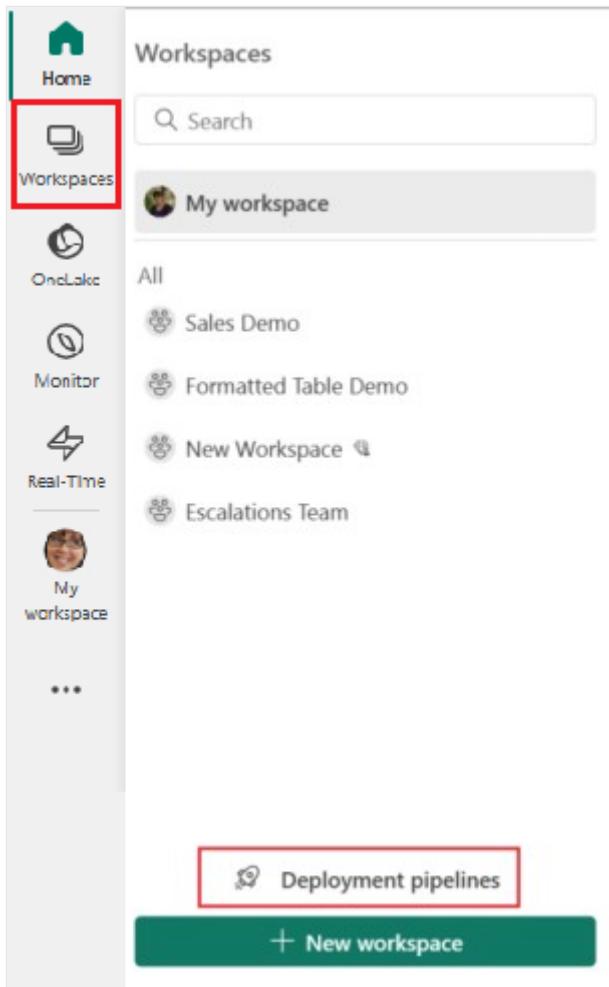
When you create a pipeline, you define how many stages it should have and what they should be called. The number of stages are permanent and can't be changed after the pipeline is created.

You can create a pipeline from the deployment pipelines entry point in Fabric (at the bottom of the workspace list), or from a specific workspace. If you create a pipeline from a workspace, the workspace is automatically assigned to the pipeline.

Create a pipeline from the deployment pipelines button in Fabric

To create a pipeline from anywhere in Fabric:

1. From the Workspaces flyout, select **Deployment pipelines**.

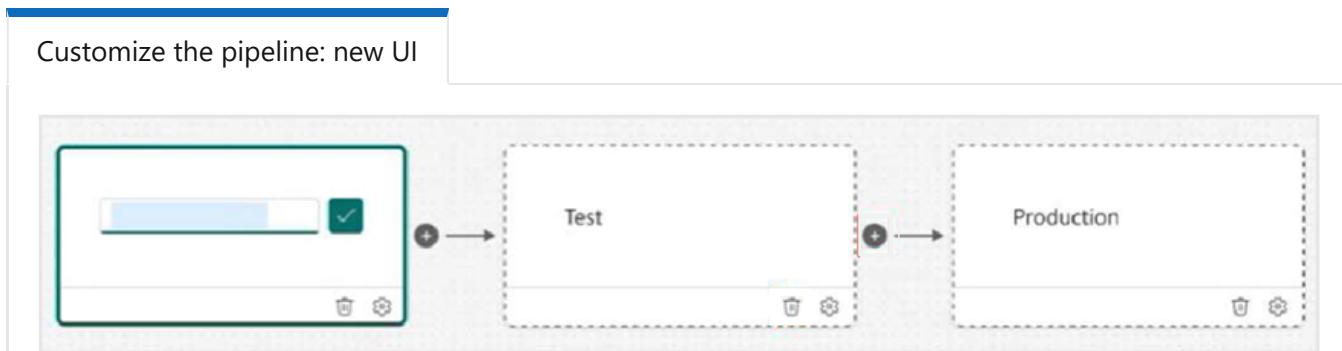


2. Select **Create pipeline**, or + **New pipeline**.

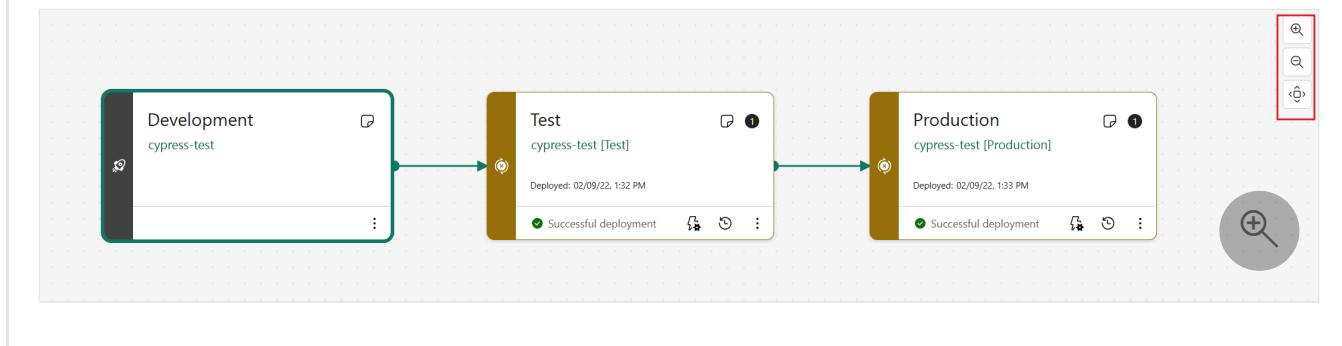
Step 2 - Name the pipeline and assign stages

1. In the *Create deployment pipeline* dialog box, enter a name and description for the pipeline, and select **Next**.
2. Set your deployment pipeline's structure by defining the required stages for your deployment pipeline. By default, the pipeline has three stages named **Development**, **Test**, and **Production**. You can accept these default stages or change the number of stages and

their names. You can have anywhere between 2-10 stages in a pipeline. You can add another stage, delete stages, or rename them by typing a new name in the box. Select **Create** (or **Create and continue**) when you're done.



To navigate between stages, zoom in and out with your mouse wheel or use the buttons in the top right. You can also drag the pipeline with your mouse to move it around.



After the pipeline is created, you can share it with other users, edit, or delete it. When you share a pipeline with others, they receive access to the pipeline and become [pipeline admins](#). Pipeline access enables users to view, share, edit, and delete the pipeline.

Step 3 - Assign a workspace

! Note

If you're creating a pipeline directly from a workspace, you can skip this stage as the workspace is already selected.

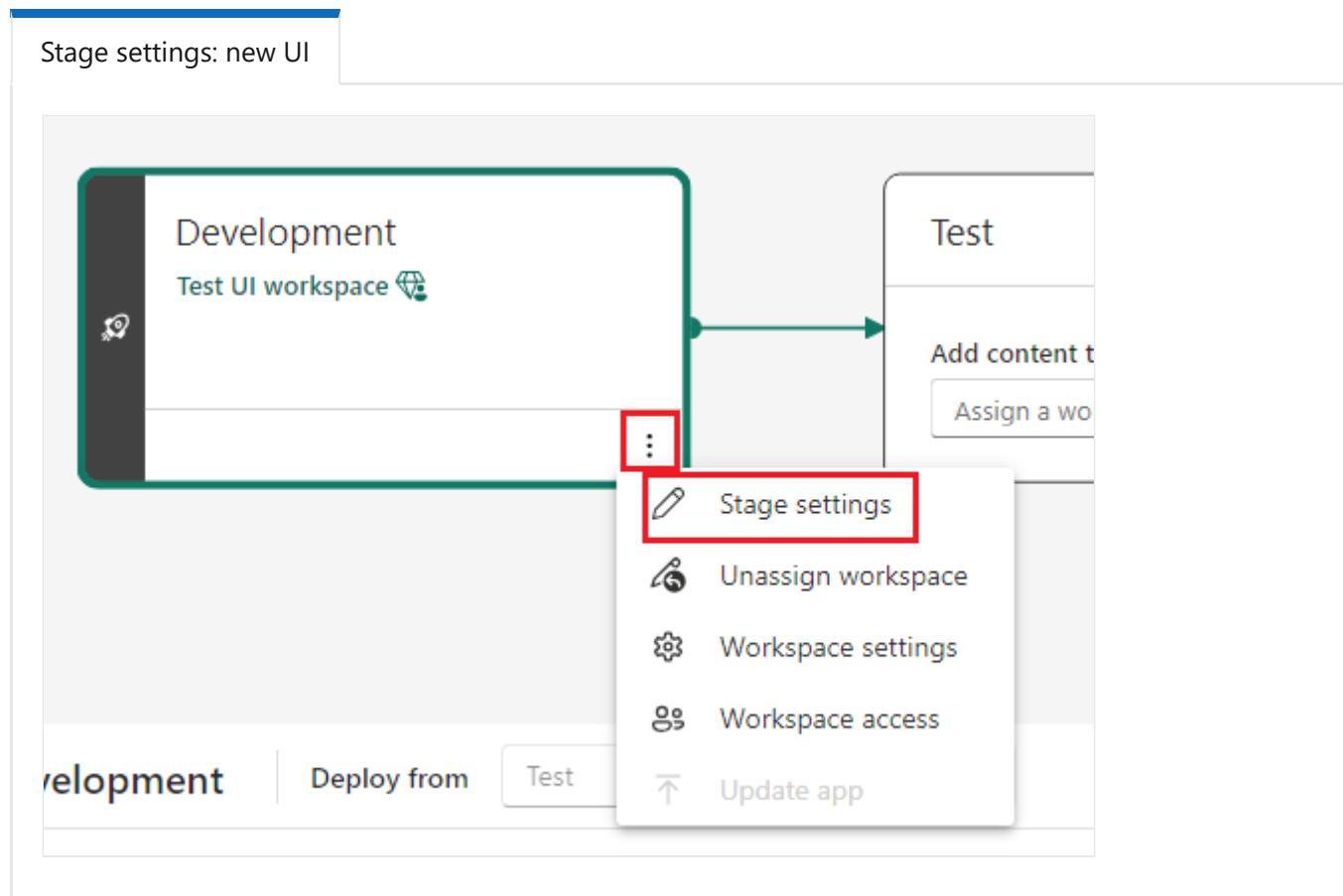
After creating a pipeline, you need to add the content you want to manage to the pipeline. Adding content to the pipeline is done by assigning a workspace to the pipeline stage. You can assign a workspace to any stage.

Follow the instructions in the link to [assign a workspace to a pipeline](#).

Step 4 - Make a stage public (optional)

By default, the final stage of the pipeline is made public. A consumer of a public stage without access to the pipeline sees it as a regular workspace, without the stage name and deployment pipeline icon on the workspace page next to the workspace name.

You can have as many public stages as you want, or none at all. To change the public status of a stage at any time, go to the pipeline stage settings and check or uncheck the **Make this stage public** box.



Set the **Make this stage public** box, and then save.

Stage settings

X

Stage name *

Production

Description

Your content has been tested and is ready to distribute to your consumers as an app or by access to the production workspace.

X Clear description

 Set as public 



Yes

Step 5 - Deploy to an empty stage

When you finished working with content in one pipeline stage, you can deploy it to the next stage. Deploying content to another stage is often done after you performed some actions in the pipeline. For example, made development changes to your content in the development stage, or tested your content in the test stage. A typical workflow for moving content from stage to stage, is development to test, and then test to production, but you can deploy in any direction. You can learn more about this process, in the [deploy content to an existing workspace](#) section.

Deployment pipelines offer three options for deploying your content:

- [Full deployment](#) - Deploy all your content to the target stage.

- **Selective deployment** - Select which content to deploy to the target stage.
- Backward deployment - Deploy content from a later stage to an earlier stage in the pipeline. Currently, backward deployment is only possible when the target stage is empty (has no workspace assigned to it).

After you choose how to deploy your content, you can [Review your deployment and leave a note](#).

Step 6 - Deploy content from one stage to another

Once you have content in a pipeline stage, you can deploy it to the next stage, even if the next stage workspace has content. [Paired items](#) are overwritten. You can learn more about this process, in the [deploy content to an existing workspace](#) section.

You can review the deployment history to see the last time content was deployed to each stage.

Deployment history is useful for establishing when a stage was last updated. It can also be helpful if you want to track time between deployments.

To examine the differences between the two pipelines before you deploy, see [compare content in different deployment stages](#).

Step 7 - Create deployment rules (optional)

When you're working in a deployment pipeline, different stages can have different configurations. For example, each stage can have different databases or different query parameters. The development stage might query sample data from the database, while the test and production stages query the entire database.

When you deploy content between pipeline stages, configuring deployment rules enables you to allow changes to content while keeping some settings intact. For example, you can define a rule for a semantic model in a production stage to point to a production database. Define the rule in the production stage under the appropriate semantic model. Once a rule is defined or changed, redeploy the content. The deployed content inherits the value defined in the deployment rule, and always applies as long as the rule is unchanged and valid.

[Read about how to define deployment rules.](#)

Related content

- Assign a workspace to a pipeline stage
 - Troubleshooting deployment pipelines
-

Last updated on 12/15/2025

The deployment pipelines process

The deployment process lets you clone content from one stage in the deployment pipeline to another, typically from development to test, and from test to production.

During deployment, Microsoft Fabric copies the content from the source stage to the target stage. The connections between the copied items are kept during the copy process. Fabric also applies the configured deployment rules to the updated content in the target stage. Deploying content might take a while, depending on the number of items being deployed. During this time, you can navigate to other pages in the portal, but you can't use the content in the target stage.

You can also deploy content programmatically, using the [deployment pipelines REST APIs](#). You can learn more about this process in [Automate your deployment pipeline using APIs and DevOps](#).

 **Note**

The new Deployment pipeline user interface is currently in preview. To turn on or use the new UI, see [Begin using the new UI](#).

There are two main parts of the deployment pipelines process:

- [Define the deployment pipeline structure](#)
- [Add content to the stages](#)

Define the deployment pipeline structure

When you create a pipeline, you define how many stages you want and what they should be called. You can also make one or more stages public. The number of stages and their names are permanent, and can't be changed after the pipeline is created. However, you can change the public status of a stage at any time.

To define a pipeline, follow the instructions in [Create a deployment pipeline](#).

Add content to the stages

You can add content to a pipeline stage in two ways:

- [Assigning a workspace to an empty stage](#)
- [Deploying content from one stage to another](#)

Assign a workspace to an empty stage

When you assign content to an empty stage, a new workspace is created on a capacity for the stage you deploy to. All the metadata in the reports, dashboards, and semantic models of the original workspace is copied to the new workspace in the stage you're deploying to.

After the deployment is complete, refresh the semantic models so that you can use the newly copied content. The semantic model refresh is required because data isn't copied from one stage to another. To understand which item properties are copied during the deployment process, and which item properties aren't copied, review the [item properties copied during deployment](#) section.

For instructions on how to assign and unassign workspaces to deployment pipeline stages, see [Assign a workspace to a Microsoft Fabric deployment pipeline](#).

Create a workspace

The first time you deploy content, deployment pipelines checks if you have permissions.

If you have permissions, the content of the workspace is copied to the stage you're deploying to, and a new workspace for that stage is created on the capacity.

If you don't have permissions, the workspace is created but the content isn't copied. You can ask a capacity admin to add your workspace to a capacity, or ask for assignment permissions for the capacity. Later, when the workspace is assigned to a capacity, you can deploy content to this workspace.

If you're using [Premium Per User \(PPU\)](#), your workspace is automatically associated with your PPU. In such cases, permissions aren't required. However, if you create a workspace with a PPU, only other PPU users can access it. In addition, only PPU users can consume content created in such workspaces.

Workspace and content ownership

The deploying user automatically becomes the owner of the cloned semantic models, and the only admin of the new workspace.

Deploy content from one stage to another

There are several ways to deploy content from one stage to another. You can deploy all the content, or you can [select which items to deploy](#).

You can deploy content to any adjacent stage, in either direction.

Deploying content from a working production pipeline to a stage that has an existing workspace, includes the following steps:

- Deploying new content as an addition to the content already there.
- Deploying updated content to replace some of the content already there.

After the initial deployment:

- the gateway associated with the target item is not automatically mapped to the corresponding data source.
- You need to manually configure this mapping through the target item's settings page
- After configuring verify that the data refresh completes successfully.
- Subsequent deployments do not alter or reset this gateway configuration.

Deployment process

When content from the source stage is copied to the target stage, Fabric identifies existing content in the target stage and overwrites it. To identify which content item needs to be overwritten, deployment pipelines uses the connection between the parent item and its clones. This connection is kept when new content is created. The overwrite operation only overwrites the content of the item. The item's ID, URL, and permissions remain unchanged.

In the target stage, [item properties that aren't copied](#), remain as they were before deployment. New content and new items are copied from the source stage to the target stage.

Autobinding

In Fabric, when items are connected, one of the items depends on the other. For example, a report always depends on the semantic model connected to it. A semantic model can depend on another semantic model, and can also be connected to several reports that depend on it. If there's a connection between two items, deployment pipelines always tries to maintain this connection.

Autobinding in the same workspace

During deployment, deployment pipelines checks for dependencies. The deployment either succeeds or fails, depending on the location of the item that provides the data that the deployed item depends on.

- **Linked item exists in the target stage** - Deployment pipelines automatically connect (autobind) the deployed item to the item it depends on in the deployed stage. For example, if you deploy a paginated report from development to test, and the report is connected to a semantic model that was previously deployed to the test stage, it automatically connects to the semantic model in the test stage.
- **Linked item doesn't exist in the target stage** - Deployment pipelines fail a deployment if an item has a dependency on another item, and the item providing the data isn't deployed and doesn't reside in the target stage. For example, if you deploy a report from development to test, and the test stage doesn't contain its semantic model, the deployment fails. To avoid failed deployments due to dependent items not being deployed, use the *Select related* button. *Select related* automatically selects all the related items that provide dependencies to the items you're about to deploy.

Autobinding works only with items that are supported by deployment pipelines and reside within Fabric. To view the dependencies of an item, from the item's *More options* menu, select *View lineage*.

New View lineage UI

The screenshot shows a deployment pipeline interface with a toolbar at the top. The 'Test' stage is selected. The 'Deploy from' dropdown is set to 'Development'. A green 'Deploy' button has a notification badge with the number '1'. Below the toolbar is a table listing items in the 'Selected stage item' column and their 'Type' in the 'Type' column. One item, 'V3-SqlAzure-DQ', is checked. A context menu is open over this item, listing options: Dataflow, Analyze in Excel, Delete, Quick insights, Save a copy, Settings, View usage metrics report, View workspace lineage, **View item lineage** (which is highlighted with a red box), Create paginated report, Manage permissions, Move to, and Ask Copilot.

Selected stage item	Type
NewEmail2	Dataflow
SqlAzureDataflowApp1	Analyze in Excel
V3-SqlAzure-Cached	Delete
V3-SqlAzure-Cached	Quick insights
V3-SqlAzure-Cached.pbix	Save a copy
V3-SqlAzure-DQ	Settings
V3-SqlAzure-DQ	View usage metrics report
V3-SqlAzure-DQ	View workspace lineage
V3-SqlAzure-DQ	View item lineage
V3-SqlAzure-DQ.pbix	Create paginated report
V3-SqlAzure-DQ.pbix	Manage permissions
V3-SqlAzure-DQ.pbix	Move to
V3-SqlAzure-DQ.pbix	Ask Copilot

Autobinding across workspaces

Deployment pipelines automatically binds items that are connected across pipelines, if they're in the same pipeline stage. When you deploy such items, deployment pipelines attempts to establish a new connection between the deployed item and the item connected to it in the other pipeline. For example, if you have a report in the test stage of pipeline *A* that's connected to a semantic model in the test stage of pipeline *B*, deployment pipelines recognizes this connection.

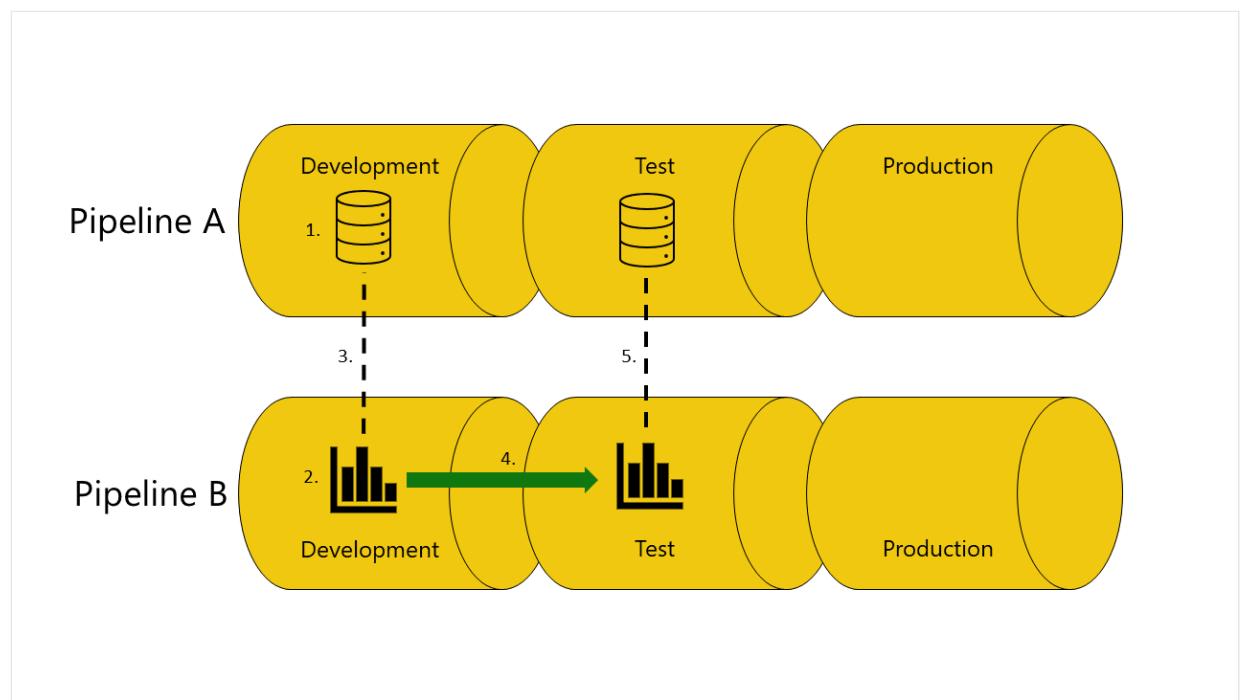
⚠ Note

Each pipeline must have the same number of stages. So for example, if pipeline *A* has 3 stages, then pipeline *B* must also have 3 stages. Pipeline *A* cannot have 3 stages and pipeline *B* 5 stages for autobinding to succeed.

Here's an example with illustrations that help demonstrate how autobinding across pipelines works:

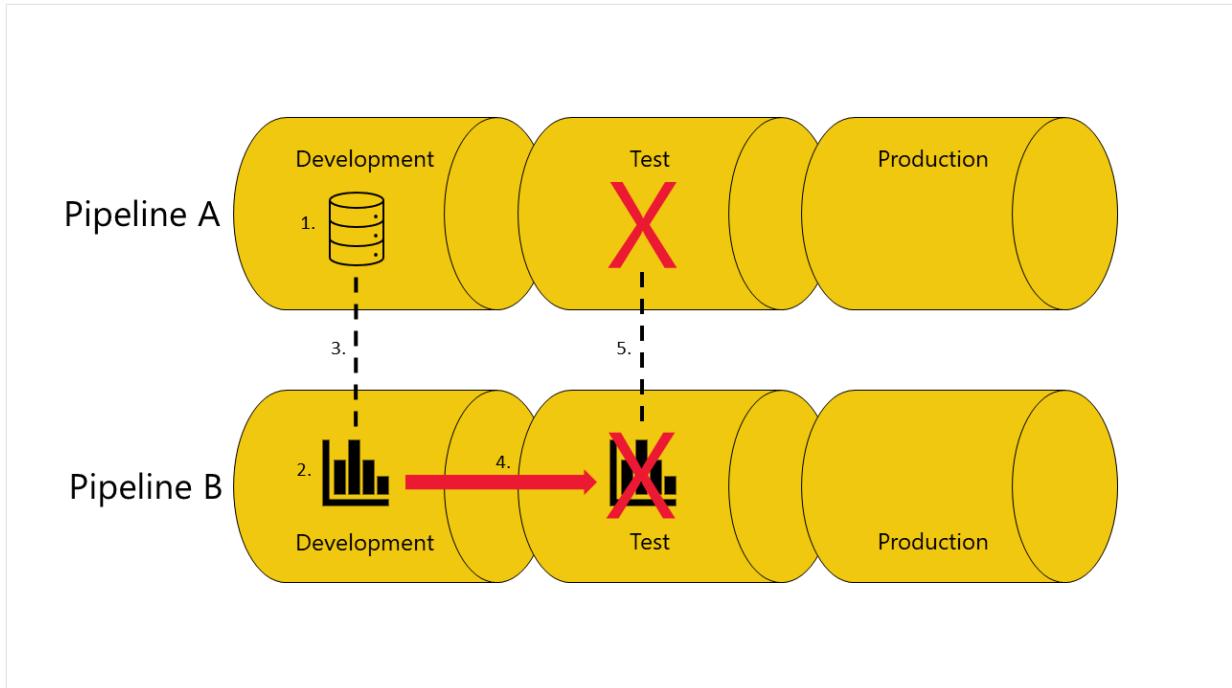
- You have a semantic model in the development stage of pipeline A.
- You also have a report in the development stage of pipeline B.
- Your report in pipeline B is connected to your semantic model in pipeline A. Your report depends on this semantic model.
- You deploy the report in pipeline B from the development stage to the test stage.
- The deployment succeeds or fails, depending on whether or not you have a copy of the semantic model it depends on in the test stage of pipeline A:
 - *If you have a copy of the semantic model the report depends on in the test stage of pipeline A:*

The deployment succeeds, and deployment pipelines connects (autobind) the report in the test stage of pipeline B, to the semantic model in the test stage of pipeline A.



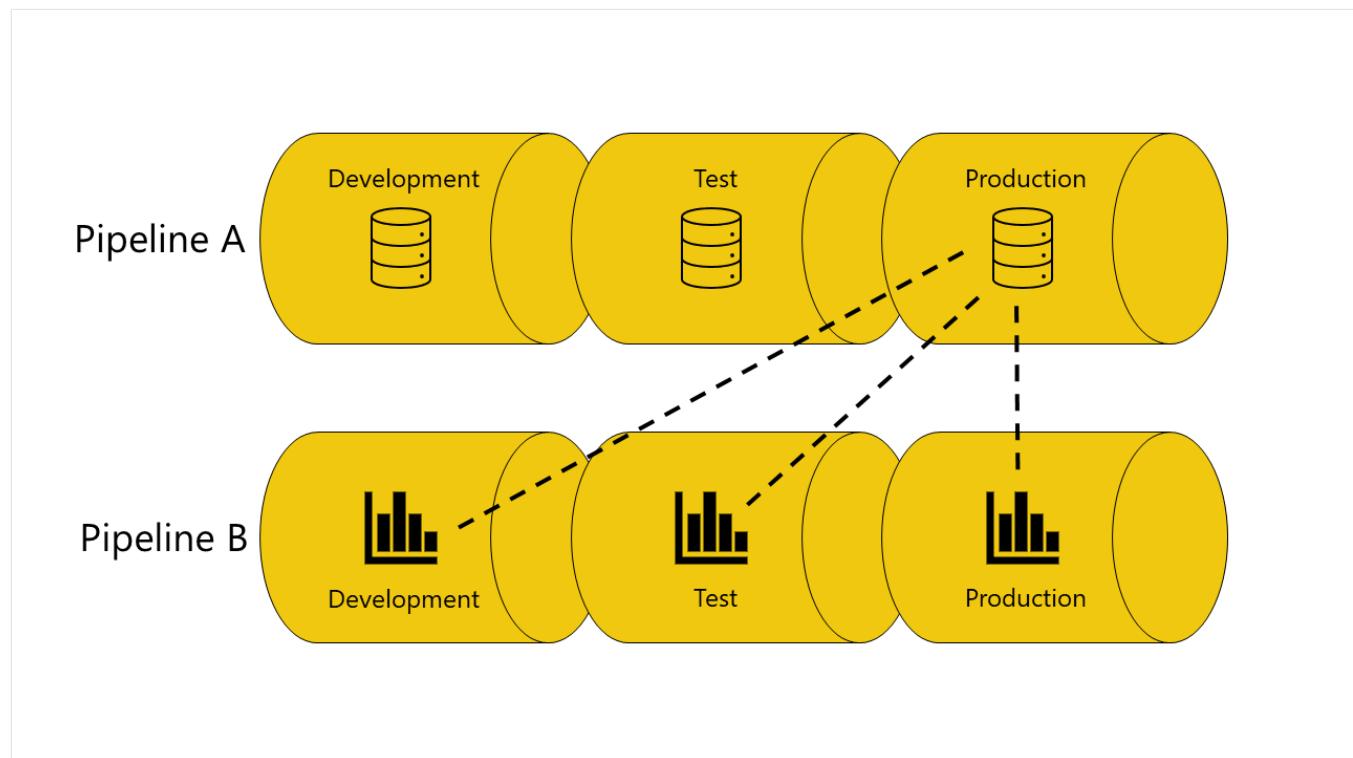
- *If you don't have a copy of the semantic model the report depends on in the test stage of pipeline A:*

The deployment fails because deployment pipelines can't connect (autobind) the report in the test stage in pipeline B, to the semantic model it depends on in the test stage of pipeline A.



Avoid using autobinding

In some cases, you might not want to use autobinding. For example, if you have one pipeline for developing organizational semantic models, and another for creating reports. In this case, you might want all the reports to always be connected to semantic models in the production stage of the pipeline they belong to. In this case, avoid using the autobinding feature.



There are three methods you can use to avoid using autobinding:

- Don't connect the item to corresponding stages. When the items aren't connected in the same stage, deployment pipelines keeps the original connection. For example, if you have

a report in the development stage of pipeline B that's connected to a semantic model in the production stage of pipeline A. When you deploy the report to the test stage of pipeline B, it remains connected to the semantic model in the production stage of pipeline A.

- Define a parameter rule. This option isn't available for reports. You can only use it with semantic models and dataflows.
- Connect your reports, dashboards, and tiles to a proxy semantic model or dataflow that isn't connected to a pipeline.

Autobinding and parameters

Parameters can be used to control the connections between semantic models or dataflows and the items that they depend on. When a parameter controls the connection, autobinding after deployment doesn't take place, even when the connection includes a parameter that applies to the semantic model's or dataflow's ID, or the workspace ID. In those cases, rebind the items after the deployment by changing the parameter value, or by using [parameter rules](#).

 Note

If you're using parameter rules to rebind items, the parameters must be of type `Text`.

Refreshing data

Data in the target item, such as a semantic model or dataflow, is kept when possible. If there are no changes to an item that holds the data, the data is kept as it was before the deployment.

In many cases, when you have a small change such as adding or removing a table, Fabric keeps the original data. For breaking schema changes, or changes in the data source connection, a full refresh is required.

Requirements for deploying to a stage with an existing workspace

Any [licensed user](#) who's a contributor of both the target and source deployment workspaces, can deploy content that resides on a [capacity](#) to a stage with an existing workspace. For more information, review the [permissions](#) section.

Folders in deployment pipelines (preview)

Folders enable users to efficiently organize and manage workspace items in a familiar way. When you deploy content that contains folders to a different stage, the folder hierarchy of the applied items is automatically applied.

! Note

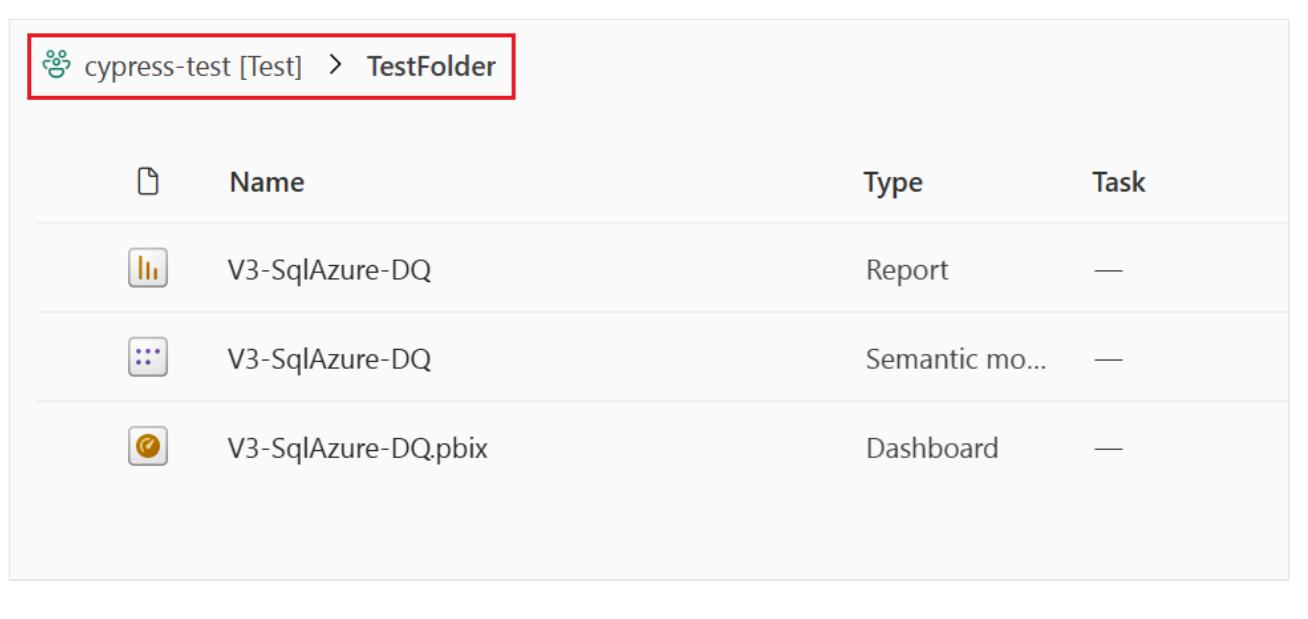
Items cannot be selected for deployment across workspace folders in the default stage view. However switching to [flat list view](#) allows you to select items for deployment across workspace folders.

Folders representation

New folders representation UI

The workspace content is shown as it's structured in the workspace. Folders are listed, and in order to see their items you need to select the folder. An item's full path is shown at the top of the items list. Since a deployment is of items only, you can only select a folder that contains supported items. Selecting a folder for deployment means selecting all its items and subfolders with their items for a deployment.

This picture shows the contents of a folder inside the workspace. The full pathname of the folder is shown at the top of the list.



cypress-test [Test] > TestFolder

 Name	Type	Task
 V3-SqlAzure-DQ	Report	—
 V3-SqlAzure-DQ	Semantic mo...	—
 V3-SqlAzure-DQ.pbix	Dashboard	—

In Deployment pipelines, folders are considered part of an item's name (an item name includes its full path). When an item is deployed, after its path was changed (moved from folder A to folder B, for example), then Deployment pipelines applies this change to its paired item during

deployment - the paired item will be moved as well to folder B. If folder B doesn't exist in the stage we're deploying to, it's created in its workspace first. Folders can be seen and managed only on the workspace page.

With the current view of the folders hierarchy, you can select for deployment, only items in the same folder level. You cannot select items across folders.

Flat list view of deployment pipelines allows you to select items regardless of its location. With the flat list view, you can select items across folders, regarding their location in the workspace. For more information, see [flat list view](#).

Identify items that were moved to different folders

Since folders are considered part of the item's name, items moved into a different folder in the workspace, are identified on Deployment pipelines page as *Different* when compared. This item doesn't appear in the compare window since it isn't a schema change but settings change.

Moved folder item in new UI				
	Selected stage item	Type	Compared to source	Source stage item
<input type="checkbox"/>	NewEmail2	...	Dataflow	Different from source  /TestFolder/NewEmail2
<input type="checkbox"/>	SqlAzureDataflowApp1		Dataflow	Same as source  SqlAzureDataflowApp1

- Individual folders can't be deployed manually in deployment pipelines. Their deployment is triggered automatically when at least one of their items is deployed.
- The folder hierarchy of paired items is updated only during deployment. During assignment, after the pairing process, the hierarchy of paired items isn't updated yet.
- Since a folder is deployed only if one of its items is deployed, an empty folder can't be deployed.
- Deploying one item out of several in a folder also updates the structure of the items that aren't deployed in the target stage even though the items themselves aren't deployed.

Parent-child item representation

Parent child relationships only appear in the new UI. They looks the same as in the workspace. The child isn't deployed but recreated on the target stage

Test	Deploy from	Development	Deploy	
	Selected stage item	Type	Compared to source	Source stage item
	Folder A	Folder	—	—
	Corporate Spend	Report	Same as source	Corporate Spend
	Corporate Spend	Semantic model	Same as source	Corporate Spend
	Env-Small	Environment (Preview)	Not in source	—
	Environment01	Environment (Preview)	Not in source	—
	ExpandReport	Semantic model (default)	Unsupported	—
	ExpandReport	Lakehouse (Preview)	Same as source	ExpandReport
	ExpandReport	SQL analytics endpoint	—	—
	Notebook 1	Notebook (Preview)	Not in source	—

Item properties copied during deployment

For a list of supported items, see [Deployment pipelines supported items](#).

During deployment, the following item properties are copied and overwrite the item properties at the target stage:

- Data sources ([deployment rules](#) are supported)
- Parameters ([deployment rules](#) are supported)
- Report visuals
- Report pages
- Dashboard tiles
- Model metadata
- Item relationships
- **Sensitivity labels** are copied *only* when one of the following conditions is met. If these conditions aren't met, sensitivity labels *aren't* copied during deployment.
 - A new item is deployed, or an existing item is deployed to an empty stage.

! Note

In cases where default labeling is enabled on the tenant, and the default label is valid, if the item being deployed is a semantic model or dataflow, the label is copied from the source item **only** if the label has protection. If the label isn't protected, the default label is applied to the newly created target semantic model or dataflow.

- The source item has a label with protection and the target item doesn't. In this case, a pop-up window asks for consent to override the target sensitivity label.

See also [Data loss prevention \(DLP\) considerations](#).

Item properties that are not copied

The following item properties aren't copied during deployment:

- Data - Data isn't copied. Only metadata is copied
- URL
- ID
- Permissions - For a workspace or a specific item
- Workspace settings - Each stage has its own workspace
- App content and settings - To update your apps, see [Update content to Power BI apps](#)
- [Personal bookmarks](#)

The following semantic model properties are also not copied during deployment:

- Role assignment
- Refresh schedule
- Data source credentials
- Query caching settings (can be inherited from the capacity)
- Endorsement settings

Supported semantic model features

Deployment pipelines supports many semantic model features. This section lists two semantic model features that can enhance your deployment pipelines experience:

- [Incremental refresh](#)
- [Composite models](#)

Incremental refresh

Deployment pipelines supports [incremental refresh](#), a feature that allows large semantic models faster and more reliable refreshes, with lower consumption.

With deployment pipelines, you can make updates to a semantic model with incremental refresh while retaining both data and partitions. When you deploy the semantic model, the policy is copied along.

To understand how incremental refresh behaves with dataflows, see [why do I see two data sources connected to my dataflow after using dataflow rules?](#)

 **Note**

Incremental refresh settings aren't copied in Gen 1.

Activating incremental refresh in a pipeline

To enable incremental refresh, [configure it in Power BI Desktop](#), and then publish your semantic model. After you publish, the incremental refresh policy is similar across the pipeline, and can be authored only in Power BI Desktop.

Once your pipeline is configured with incremental refresh, we recommend that you use the following flow:

1. Make changes to your `.pbix` file in Power BI Desktop. To avoid long waiting times, you can make changes using a sample of your data.
2. Upload your `.pbix` file to the first (usually *development*) stage.
3. Deploy your content to the next stage. After deployment, the changes you made will apply to the entire semantic model you're using.
4. Review the changes you made in each stage, and after you verify them, deploy to the next stage until you get to the final stage.

Usage examples

The following are a few examples of how you can integrate incremental refresh with deployment pipelines.

- [Create a new pipeline](#) and connect it to a workspace with a semantic model that has incremental refresh enabled.
- Enable incremental refresh in a semantic model that's already in a *development* workspace.
- Create a pipeline from a production workspace that has a semantic model that uses incremental refresh. For example, assign the workspace to a new pipeline's *production* stage, and use backwards deployment to deploy to the *test* stage, and then to the *development* stage.
- Publish a semantic model that uses incremental refresh to a workspace that's part of an existing pipeline.

Incremental refresh limitations

For incremental refresh, deployment pipelines only supports semantic models that use [enhanced semantic model metadata](#). All semantic models created or modified with Power BI Desktop automatically implement enhanced semantic model metadata.

When republishing a semantic model to an active pipeline with incremental refresh enabled, the following changes result in deployment failure due to data loss potential:

- Republishing a semantic model that doesn't use incremental refresh, to replace a semantic model that has incremental refresh enabled.
- Renaming a table that has incremental refresh enabled.
- Renaming noncalculated columns in a table with incremental refresh enabled.

Other changes such as adding a column, removing a column, and renaming a calculated column, are permitted. However, if the changes affect the display, you need to refresh before the change is visible.

Composite models

Using [composite models](#) you can set up a report with multiple data connections.

You can use the composite models functionality to connect a Fabric semantic model to an external semantic model such as Azure Analysis Services. For more information, see [Using DirectQuery for Fabric semantic models and Azure Analysis Services](#).

In a deployment pipeline, you can use composite models to connect a semantic model to another Fabric semantic model external to the pipeline.

Automatic aggregations

[Automatic aggregations](#) are built on top of user defined aggregations and use machine learning to continuously optimize DirectQuery semantic models for maximum report query performance.

Each semantic model keeps its automatic aggregations after deployment. Deployment pipelines doesn't change a semantic model's automatic aggregation. This means that if you deploy a semantic model with an automatic aggregation, the automatic aggregation in the target stage remains as is, and isn't overwritten by the automatic aggregation deployed from the source stage.

To enable automatic aggregations, follow the instructions in [configure the automatic aggregation](#).

Hybrid tables

Hybrid tables are tables with [incremental refresh](#) that can have both import and direct query partitions. During a clean deployment, both the refresh policy and the hybrid table partitions are copied. When you're deploying to a pipeline stage that already has hybrid table partitions, only the refresh policy is copied. To update the partitions, refresh the table.

Update content to Power BI apps

[Power BI apps](#) are the recommended way of distributing content to free Fabric consumers. You can update the content of your Power BI apps using a deployment pipeline, giving you more control and flexibility when it comes to your app's lifecycle.

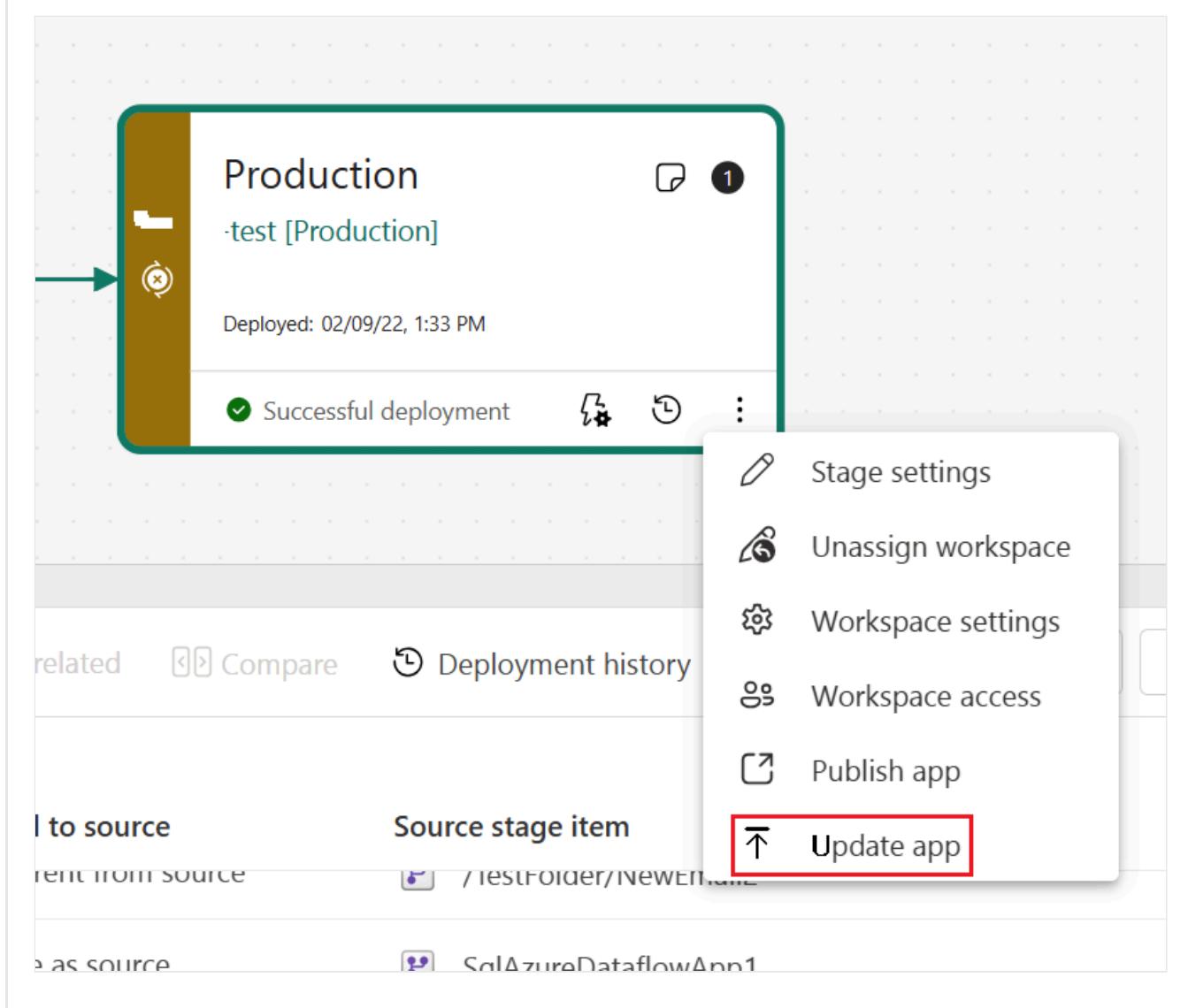
Create an app for each deployment pipeline stage, so that you can test each update from an end user's point of view. Use the **publish** or **view** button in the workspace card to publish or view the app in a specific pipeline stage.

Publish app - new UI

The screenshot shows the Microsoft Fabric app page for a production stage. The main card displays the stage name "Production", a deployment log entry "-test [Production]", and a deployment status message "Successful deployment". A green arrow points to the "Production" stage card. A context menu is open on the right side of the card, listing several options: "Stage settings", "Unassign workspace", "Workspace settings", "Workspace access", "Publish app" (which is highlighted with a red box), and "Update app". Below the main card, there are sections for "related", "Compare", and "Deployment history". At the bottom, there are two source items: "Source stage item" (from / testFolder/NEWEn... with a "Get from source" button) and "as source" (with a "Get as source" button).

In the production stage, you can also update the app page in Fabric, so that any content updates become available to app users.

Update app - new UI



i Important

The deployment process doesn't include updating the app content or settings. To apply changes to content or settings, you need to manually update the app in the required pipeline stage.

Permissions

Permissions are required for the pipeline, and for the workspaces that are assigned to it. Pipeline permissions and workspace permissions are granted and managed separately.

- Pipelines only have one permission, *Admin*, which is required for sharing, editing and deleting a pipeline.
- Workspaces have different permissions, also called **roles**. Workspace roles determine the level of access to a workspace in a pipeline.

- Deployment pipelines doesn't support [Microsoft 365 groups](#) as pipeline admins.

To deploy from one stage to another in the pipeline, you must be a pipeline admin, and either a contributor, member, or admin of the workspaces assigned to the stages involved. For example, a pipeline admin that isn't assigned a workspace role, can view the pipeline and share it with others. However, this user can't view the content of the workspace in the pipeline, or in the service, and can't perform deployments.

Permissions table

This section describes the deployment pipeline permissions. The permissions listed in this section might have different applications in other Fabric features.

The lowest deployment pipeline permission is *pipeline admin*, and it's required for all deployment pipeline operations.

[\[+\] Expand table](#)

User	Pipeline permissions	Comments
Pipeline admin	<ul style="list-style-type: none"> • View the pipeline • Share the pipeline with others • Edit and delete the pipeline • Unassign a workspace from a stage • Can see workspaces that are tagged as assigned to the pipeline in Power BI service 	Pipeline access doesn't grant permissions to view or take actions on the workspace content.
Workspace viewer (and pipeline admin)	<ul style="list-style-type: none"> • Consume content • Unassign a workspace from a stage 	Workspace members assigned the Viewer role without <i>build</i> permissions, can't access the semantic model or edit workspace content.
Workspace contributor (and pipeline admin)	<ul style="list-style-type: none"> • Consume content • Compare stages • View semantic models • Unassign a workspace from a stage • Deploy items (must be at least a contributor in both source and target workspaces) 	

User	Pipeline permissions	Comments
Workspace member (and pipeline admin)	<ul style="list-style-type: none"> View workspace content Compare stages Deploy items (must be at least a contributor in both source and target workspaces) Update semantic models Unassign a workspace from a stage Configure semantic model rules (you must be the semantic model owner) 	If the <i>block republish and disable package refresh</i> setting located in the tenant <i>semantic model security</i> section is enabled, only semantic model owners are able to update semantic models.
Workspace admin (and pipeline admin)	<ul style="list-style-type: none"> View workspace content Compare stages Deploy items Assign workspaces to a stage Update semantic models Unassign a workspace from a stage Configure semantic model rules (you must be the semantic model owner) 	

Granted permissions

When you're deploying Power BI items, the ownership of the deployed item might change. Review the following table to understand who can deploy each item and how the deployment affects the item's ownership.

 Expand table

Fabric Item	Required permission to deploy an existing item	Item ownership after a first time deployment	Item ownership after deployment to a stage with the item
Semantic model	Workspace member	The user who made the deployment becomes the owner	Unchanged
Dataflow	Dataflow owner	The user who made the deployment becomes the owner	Unchanged

Fabric Item	Required permission to deploy an existing item	Item ownership after a first time deployment	Item ownership after deployment to a stage with the item
Datamart	Datamart owner	The user who made the deployment becomes the owner	Unchanged
Paginated report	Workspace member	The user who made the deployment becomes the owner	The user who made the deployment becomes the owner

Required permissions for popular actions

The following table lists required permissions for popular deployment pipeline actions. Unless specified otherwise, for each action you need all the listed permissions.

 [Expand table](#)

Action	Required permissions
View the list of pipelines in your organization	No license required (free user)
Create a pipeline	A user with one of the following licenses: <ul style="list-style-type: none"> • Pro • PPU • Premium
Delete a pipeline	Pipeline admin
Add or remove a pipeline user	Pipeline admin
Assign a workspace to a stage	<ul style="list-style-type: none"> • Pipeline admin • Workspace admin (of the workspace to be assigned)
Unassign a workspace to a stage	One of the following roles: <ul style="list-style-type: none"> • Pipeline admin • Workspace admin (using the Pipelines - Unassign Workspace API)
Deploy to an empty stage (see note)	<ul style="list-style-type: none"> • Pipeline admin • Source workspace contributor

Action	Required permissions
Deploy items to the next stage (see note)	<ul style="list-style-type: none"> • Pipeline admin • Workspace contributor to both the source and target stages • To deploy datamarts or dataflows, you must be the owner of the deployed item • If the semantic model tenant admin switch is turned on and you're deploying a semantic model, you need to be the owner of the semantic model
View or set a rule	<ul style="list-style-type: none"> • Pipeline admin • Target workspace contributor, member, or admin • Owner of the item you're setting a rule for
Manage pipeline settings	Pipeline admin
View a pipeline stage	<ul style="list-style-type: none"> • Pipeline admin • Workspace reader, contributor, member, or admin. You see the items that your workspace permissions grant access to.
View the list of items in a stage	Pipeline admin
Compare two stages	<ul style="list-style-type: none"> • Pipeline admin • Workspace contributor, member, or admin for both stages
View deployment history	Pipeline admin

 **Note**

To deploy content in the GCC environment, you need to be at least a member of both the source and target workspace. Deploying as a contributor isn't supported yet.

Considerations and limitations

This section lists most of the limitations in deployment pipelines.

- [General considerations and limitations](#)
- [Semantic model limitations](#)
- [Dataflow limitations](#)
- [Datamart limitations](#)
- [Data loss prevention \(DLP\) considerations](#)

General considerations and limitations

- The workspace must reside on a [Fabric capacity](#).
- The maximum number of items that can be deployed in a single deployment is 300.
- Downloading a [.pbix](#) file after deployment isn't supported.
- [Microsoft 365 groups](#) aren't supported as pipeline admins.
- When you deploy a Power BI item for the first time, if another item in the target stage has the same name and type (for example, if both files are reports), the deployment fails.
- For a list of workspace limitations, see the [workspace assignment limitations](#).
- For a list of supported items, see [supported items](#). Any item not on the list isn't supported.
- The deployment fails if any of the items have circular or self dependencies (for example, item A references item B and item B references item A).
- PBIR reports aren't supported.

Semantic model limitations

- Datasets that use real-time data connectivity can't be deployed.
- A semantic model with DirectQuery or Composite connectivity mode that uses variation or [auto date/time](#) tables, isn't supported. For more information, see [What can I do if I have a dataset with DirectQuery or Composite connectivity mode, that uses variation or calendar tables?](#).
- During deployment, if the target semantic model is using a [live connection](#), the source semantic model must use this connection mode too.
- After deployment, downloading a semantic model (from the stage it was deployed to) isn't supported.
- For a list of deployment rule limitations, see [deployment rules limitations](#).
- If autobinding is engaged, then:
 - Native query and DirectQuery together isn't supported. This includes proxy datasets.
 - The datasource connection must be the first step in the mashup expression.
- When a Direct Lake semantic model is deployed, it doesn't automatically bind to items in the target stage. For example, if a LakeHouse is a source for a DirectLake semantic model and they're both deployed to the next stage, the DirectLake semantic model in the target stage will still be bound to the LakeHouse in the source stage. Use datasource rules to bind it to an item in the target stage. Other types of semantic models are automatically bound to the paired item in the target stage.

Dataflow limitations

- Incremental refresh settings aren't copied in Gen 1.
- When you're deploying a dataflow to an empty stage, deployment pipelines creates a new workspace and sets the dataflow storage to a Fabric blob storage. Blob storage is used even if the source workspace is configured to use Azure data lake storage Gen2 (ADLS Gen2).
- Service principal isn't supported for dataflows.
- Deploying common data model (CDM) isn't supported.
- For deployment pipeline rule limitations that affect dataflows, see [Deployment rules limitations](#).
- If a dataflow is being refreshed during deployment, the deployment fails.
- If you compare stages during a dataflow refresh, the results are unpredictable.
- Autobinding isn't supported for dataflows Gen2.

Datamart limitations

- You can't deploy a datamart with sensitivity labels.
- You need to be the datamart owner to deploy a datamart.

Data loss prevention (DLP) considerations

After deploying an item to a new stage, if you see a DLP policy tip indication on the item, try refreshing the item to see whether the indication disappears before investigating further. Because DLP runs as soon as an item is copied, possibly before other processes that bring in data or metadata (such as a default sensitivity label) have completed, DLP might have run on the item prematurely, resulting in the misapplication of the policy tip indication. Refreshing the item should cause the policy tip indication to go away.

Related content

[Get started with deployment pipelines.](#)

Assign a workspace to a Microsoft Fabric deployment pipeline

Deployment pipelines enable you to assign and unassign workspaces to any stage in a pipeline. This capability is important for organizations that already have workspaces that are used as different environments of a managed release. In such cases, you can assign each workspace to its corresponding pipeline stage, and continue working in your usual flow.

(!) Note

The new Deployment pipeline user interface is currently in [preview](#). To turn on or use the new UI, see [Begin using the new UI](#).

For more information about assigning workspaces and the implications regarding capacities and permissions, see [The deployment pipelines process](#).

Assign a workspace to any vacant pipeline stage

To assign a workspace to a deployment pipeline, the pipeline stage you want to assign the workspace to has to be vacant. To assign a workspace to a pipeline stage that already has another workspace assigned to it, [unassign](#) the current workspace from that stage and then assign the new workspace.

Before you assign a workspace to a deployment pipeline stage, review the [limitations](#) section and make sure that the workspace meets the required conditions.

(!) Note

Before you assign or unassign a workspace to a deployment pipeline, consider that every time you deploy to a vacant stage, a new workspace is created, and whenever you unassign a workspace, you lose all the stage deployment's history and configured rules.

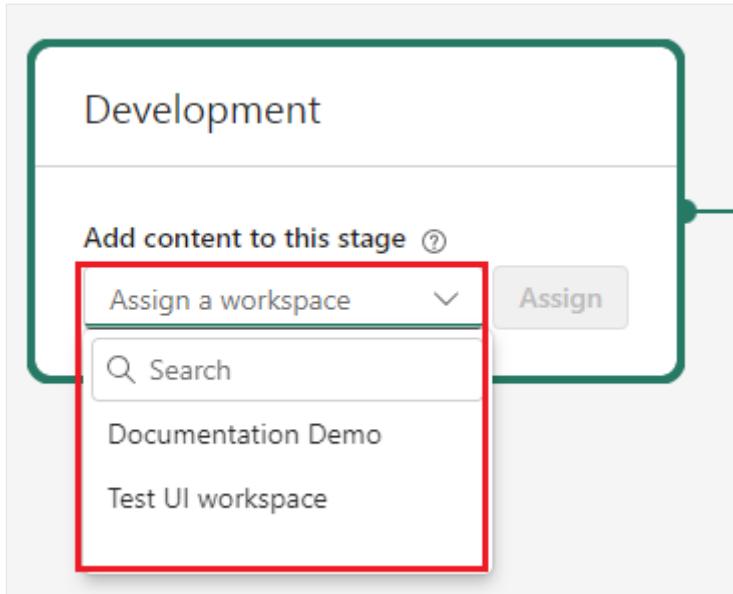
To assign a workspace to a deployment pipeline stage, follow these steps:

Assign a workspace: New UI

1. Open the deployment pipeline.

2. In the stage you want to assign a workspace to, expand the dropdown titled **Add content to this stage**.

3. Select the workspace you want to assign to this stage.



4. Select **Assign**.

Remove a workspace in a deployment pipeline

If you attempt to delete a workspace via the workspace settings, and that workspace is currently assigned to a [Deployment Pipeline](#), the deletion fails.

Delete workspace

✖ This workspace is included in a deployment pipeline. Before deleting the workspace here, you must first remove it from the pipeline. ✖

Was this error message helpful? ↳ ↘

This workspace will be permanently deleted after a retention period of 7 days, as defined by your Fabric administrator. Contact your Fabric administrator if you need to restore the workspace and recover the items in it during this retention period.

trash Remove this workspace +
🔍

To remove the workspace, you can return to the workspace page and click the **View Deployment Pipeline** button. This will direct you to the relevant pipeline, where you can unassign the workspace and then remove it.

If you do not have permissions to [unassign](#) the workspace, you will need to contact someone with administrative rights on the deployment pipeline to perform the unassignment.

Important

If the only person with access to the deployment pipeline has left the organization, then a tenant global administrator will need to use an API to release the workspace. See [How to delete a pipeline that doesn't have an owner](#).

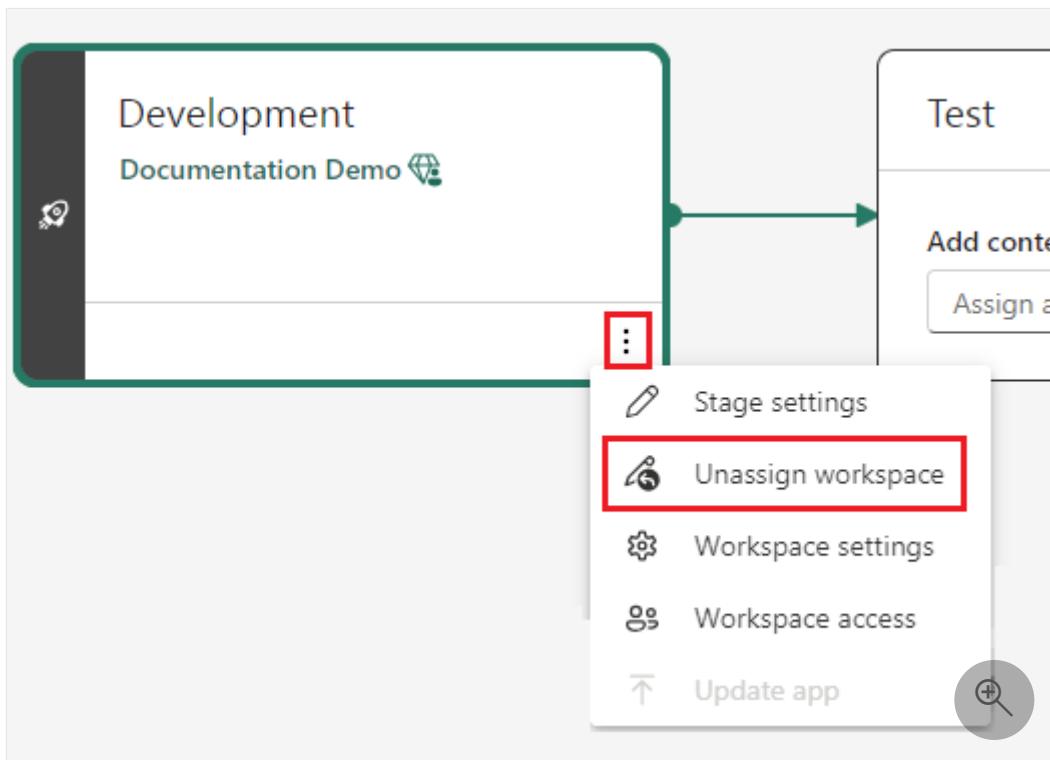
Unassign a workspace from a deployment pipeline stage

You can unassign a workspace from any deployment pipeline stage. If you want to assign a different workspace to a deployment pipeline stage, you first have to unassign the current workspace from that stage.

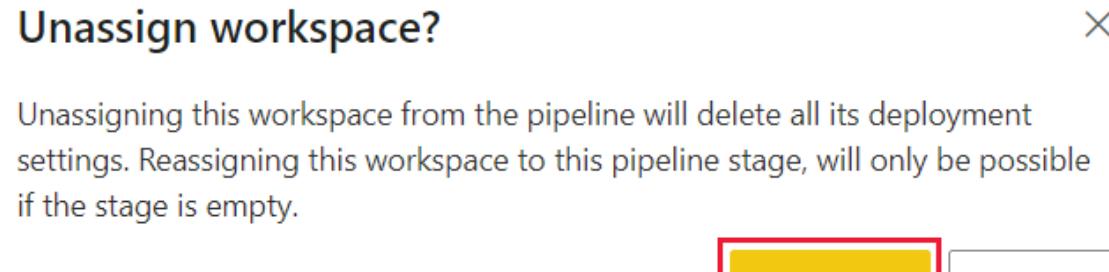
To unassign a workspace from a deployment pipeline stage, follow these steps:

Unassign a workspace: New UI

1. Open the deployment pipeline.
2. In the stage you want to unassign the workspace from, select the three dots in the lower left corner.
3. Select **Unassign workspace**.



4. In the *Unassign workspace* dialogue box, select **Unassign**.



Item pairing

Pairing is the process by which an item in one stage of the deployment pipeline is associated with the same item in the adjacent stage. If items aren't paired, even if they have the same name and type, the item in the target stage isn't overwritten during a deploy. A deploy of an unpaired item is known as a clean deploy and creates a copy of that item in the adjacent stage.

Pairing can happen in one of two ways:

- **Deployment:** when an unpaired item is copied from one stage to another using the *Deploy* button, a copy of the item is created in the next stage and paired with the item being deployed.

The following table shows when items are paired when the deploying button is used in different circumstances:

[] [Expand table](#)

Scenario	Stage A (for example, Dev)	Stage B (for example, Test)	Comment
1	Name: <i>PBI Report</i> Type: <i>Report</i>	None	Clean deploy - pairing occurs
2	Name: <i>PBI Report</i> Type: <i>Report</i>	Name: <i>PBI Report</i> Type: <i>Report</i>	If items are paired, then pressing deploy overwrites stage B.
3	Name: <i>PBI Report</i>	Name: <i>PBI Report</i>	If items aren't paired, the report in stage A is copied to stage B. There are then two files in stage B with the

Scenario	Stage A (for example, Dev)	Stage B (for example, Test)	Comment
	Type: Report	Type: Report	same name- one paired and one unpaired. Deployments continues to succeed between the paired items.

- **Assigning a workspace to a deployment stage:** when a workspace is assigned to a deployment stage the deployment pipeline attempts to pair items. The pairing criteria are:
 - Item Name
 - Item Type
 - Folder Location (used as a tie breaker when a stage contains duplicate items (two or more items with the same name and type))

If a single item in each stage has the same name and type, then pairing occurs. If there's more than one item in a stage that has the same name and type, then items are paired if they're in the same folder. If the folders aren't the same, pairing fails.

The following table shows when items are paired when a workspace is assigned in different circumstances:

[Expand table](#)

Scenario	Stage A (for example, Dev)	Stage B (for example, Test)	Comment
1	Name: PBI Report Type: Report	Name: PBI Report Type: Report	✓ Pairing occurs
2	Name: PBI Report Type: Report	Name: PBI Report Type: Report	✗ Pairing doesn't occur (duplicates). ✗ Deployment fails.
		Name: PBI Report Type: Report	✗ Pairing doesn't occur (duplicates). ✗ Deployment fails.
3	Name: PBI Report Type: Report Folder A	Name: PBI Report Type: Report Folder B	✓ Deployment succeeds but ✗ this report isn't paired with dev
		Name: PBI Report Type: Report Folder A	✓ Pairing occurs using folder as a tie breaker for duplicates
		Name: PBI Report Type: Report No folder	✓ Deployment succeeds but ✗ this report isn't paired with dev

⚠ Note

Once items are paired, renaming them *doesn't* unpair the items. Thus, there can be paired items with different names.

See which items are paired

Paired items appear on the same line in the pipeline content list. Items that aren't paired, appear on a line by themselves:

The screenshot shows a user interface titled "Paired items: New UI". At the top, there are buttons for "Test", "Deploy from Development", "Deploy", and navigation links like "Select related", "Compare", "Deployment history", "Filter by keyword", and "Filter". The main area displays a table of items:

Selected stage item	Type	Compared to source	Source stage item
CloudRLS v3	Report	Same as source	CloudRLS v3
CloudRLS333	Semantic model	Same as source	CloudRLS333
full_v3	Report	Not in source	—
full_v3	Semantic model	Same as source	full_v3
—	Semantic model	Only in source	FamilyBM2M

Create nonpaired items with the same name

There's no way to manually pair items except by following the pairing rules described in the [previous section](#). Adding a new item to a workspace that's part of a pipeline, doesn't automatically pair it to an identical item in an adjacent stage. Thus, you can have identical items with the same name in adjacent workspaces that aren't paired.

Here's an example of items that were added to the directly to the *Test* workspace after it was assigned and therefore not paired with the identical item in the *Dev* pipeline:

MF Deployment Pipeline Dev

- Warehouses (Preview) 0
- environments (Preview) 0
- Lakehouses (Preview) 0
- Data pipelines (Preview) 0

Test File 1, Test File 2, Test File 1, Test File 2, Test File 1.pbix

Show less ^ Publish app Deploy

MF Deployment Pipeline Test

- Warehouses (Preview) 0
- environments (Preview) 0
- Lakehouses (Preview) 0
- Data pipelines (Preview) 0

Test File 1, Test File 1

Show less ^ Publish app Deploy

Multiple items with the same name and type in a workspace

If two or more items in the workspace to be paired have the same name, type and folder, pairing fails. Move one item to a different folder or change its name so that there are no longer two items that match an existing item in the other stage.

MF Deployment Pipeline Dev

- Warehouses (Preview) 0
- environments (Preview) 0
- Lakehouses (Preview) 0
- Data pipelines (Preview) 0

Show more ^ Publish app Deploy

⚠ Can't assign the workspace

The full path for items from the same type can't be identical. Update each duplicate item by changing the item name or by moving to a new subfolder. Then, reassign the workspace. [Learn more](#)

Test File 1 (Test)
Test File 1 (Test)

Please check the technical details for more information. If you contact support, please provide these details.

See details ^ Close

Considerations and limitations

- Only workspaces that can be assigned to a deployment pipeline appear in the dropdown list. A workspace can be assigned to a deployment pipeline stage if the following conditions apply:
 - You're an admin of the workspace.
 - The workspace isn't assigned to any other deployment pipeline.
 - The workspace resides on a [Fabric capacity](#).
 - You have at least [workspace contributor](#) permissions for the workspaces in its adjacent stages. For more information, see [Why am I getting the *workspace contributor permissions needed* error message when I try to assign a workspace?](#)
 - The workspace doesn't contain [Power BI samples](#).
 - The workspace isn't a [template app](#) workspace.
 - The workspace doesn't have a template app installed.
- When a Direct Lake semantic model is deployed, it doesn't automatically bind to items in the target stage. For example, if a LakeHouse is a source for a DirectLake semantic model and they're both deployed to the next stage, the DirectLake semantic model in the target stage will be bound to the LakeHouse in the source stage. Use datasource rules to bind it to an item in the target stage. Other types of semantic models are automatically bound to the paired item in the target stage.

Related content

[Compare content in different stages](#).

Last updated on 12/15/2025

Compare content in different deployment stages

Before you deploy content to a different stage, it can be helpful to see the differences between the two stages. The deployment pipeline home page compares consecutive deployment stages and indicates if there are any differences between them.

Deployment pipelines pairs items of two neighboring stages by combining item type and item name, to know which items to compare and to override. Items of the same name and type are paired. If there's more than one item in a workspace with the same name and type, then the items are paired if their paths are the same. If the path isn't the same, the items aren't paired. The pairing is created only once, during the first deployment of one stage to another, or during assignment of a workspace. On subsequent deployments, each deployed item overrides its paired item metadata, including its name, if it was changed. For more information on pairing, see [pairing items](#).

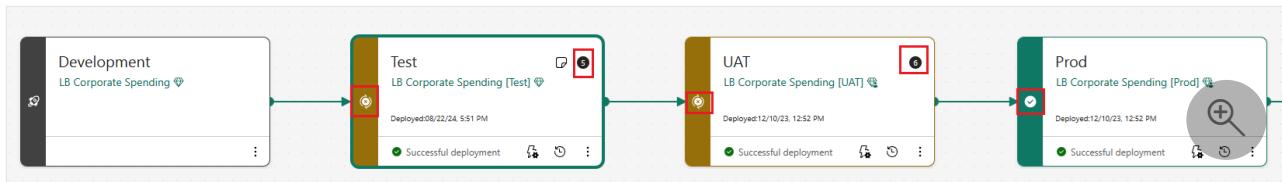
! Note

The new Deployment pipeline user interface is currently in preview. To turn on or use the new UI, see [Begin using the new UI](#).

Compare stages

New deployment pipeline UI

A comparison icon indicator appears on each stage card except for the first stage of the pipeline. It indicates if this stage is identical to the source stage (previous stage) to give a quick visual insight into the differences between them.



The comparison indicator has two states:

- **Green indicator** – The metadata for each content item in both stages, is the same.
- **Orange indicator** - Appears if at least one item in one of the compared stages were changed, added, or removed.

When you select a deployment pipelines stage, the items in the stage are listed and compared to the item they're linked to in the source stage.

The source stage is shown in the drop-down menu on the bottom pane and the name of the compared source item's name appears on the last column

The screenshot shows a comparison between the 'Production' stage and the 'Test' stage. The 'Deploy from' dropdown is set to 'Test'. The 'Deploy' button is visible at the top right. The main area displays a table with the following data:

Selected stage item	Type	Compared to source	Source stage item
NewEmail2	Dataflow	Not in source	—
SqlAzureDataflowApp1	Dataflow	Same as source	SqlAzureDataflowApp1
V3-SqlAzure-Cached	Report	Same as source	V3-SqlAzure-Cached
V3-SqlAzure-Cached	Semantic model	Same as source	V3-SqlAzure-Cached

In the stage display, items are arranged alphabetically by default. You can sort or filter the items to find the ones you're interested in, or you can search for a specific item. Each item has one of the following labels depending on the comparison status:

- **Not in source** – This item appears in the selected stage, but not in the source stage. This item can't be selected for deployment (since it doesn't exist in the source) and isn't impacted during a deployment.
- **Different from source** – A difference was identified between this item and its paired item in the source stage. The difference could be any of several things:
 - a schema change to one of the items
 - property change, like a name change (considering full path of folders, if any)
 - deployment rules that were set for this item but not applied yet (requires deployment of the item).

After deployment, the item in the source stage overwrites the item in the target stage, regardless of where the change was made.

- **Only in source** – A new item identified in the source stage. This item doesn't exist in the selected stage and therefore serves as a placeholder with no item name in the first column (under *name*). After deployment, this item will be cloned to this stage.
- **Same as source** – No difference was identified between this item and its pair in the source stage.

!**Note**

- If you make changes to a folder, such as moving its location or renaming it, even if you didn't change the items in it, the items are still treated as if you renamed them.

Therefore, when comparing pipelines the items are labeled as *Different*.

- Deployment will not impact items not in the source stage.

Review changes to paired items

If there's a change in the schema, you can see the differences between the two items by selecting the **Compare** button.

New Deployment change review button

To compare the items in the two stages, select the **Compare** icon:



A pop-up window opens with a line by line comparison of the item's content as it [currently looks in the two stages being compared](#).

The top of the screen has the following information:

1. The workspace name followed by name of the item as it appears in the source (*to be deployed*) stage.
2. The total number of changes made to the file in the *to be modified* stage (green) and the *to be deployed* stage (red).
3. Up and down arrows that take you to the previous or next difference in the file.
4. A navigation bar on the right side with red or green bars highlighting where the changes are in the file.
5. Buttons that toggle between a side-by-side view and an inline view of the changes.

Side-by-side view

A line by line comparison of the items. On the left is the schema of this stage's item. On the right is the schema of the paired item in the source stage.

```

1 {
2   "annotations": [
3     {
4       "name": "PBIDesktopVersion",
5       "value": "2.91.884.0 (21.03)"
6     },
7     {
8       "name": "PBI_QueryOrder",
9       "value": "[\"Store\", \"Sales\", \"Item\", \"Time\", \"District\"]"
10    },
11   {
12     "name": "__PBI_TimeIntelligenceEnabled",
13     "value": "1"
14   },
15 ],
16 "culture": "en-US",

```

```

1 {
2   "annotations": [
3     {
4       "name": "PBIDesktopVersion",
5       "value": "2.115.6858.1"
6     },
7     {
8       "name": "PBI_QueryOrder",
9       "value": "[\"Store\", \"Sales\", \"Item\", \"Time\", \"District\"]"
10    },
11   {
12     "name": "__PBI_TimeIntelligenceEnabled",
13     "value": "1"
14   },
15 ],
16 "culture": "en-US",

```

Compare changes

In the *side-by-side* comparison view of the items, the code area is split in two:

- On the **left** is the item's content in the *target* stage of the deployment. This stage will be modified at the next deployment. Its content will be overridden.
- On the **right** is the item's content in the *source* stage of the deployment. This stage will be deployed. Its content will be applied.
- The lines on each side appear in the same order, so each line is next to its equivalent in the compared stage.

The *inline* comparison view, as opposed to the side-by-side view, shows each line in the *target* (to be modified) stage underneath its equivalent in the *source* (To be deployed) stage.

In both comparison displays, whether inline or side-by-side, the differences are highlighted as follows:

- The file content lines are numbered and the lines that were changed are marked as follows:
 - Changes shown in the *To be modified* stage will be removed or overwritten during the next deployment. They're highlighted in **red** with a '-' sign next to the number.
 - Changes shown in the *To be deployed* stage are the new values that will be applied during the next deployment. They're highlighted in **green** with a '+' sign next to the number.
- In the modified lines, the specific characters that were added or deleted are highlighted in a darker shade.

File modifications before comparison

Both versions of the content shown in the *Compare* window are modified in the following ways to make the comparison easier:

- Data source and parameter rules are applied to the source item so that the data source you see is the one that's deployed.
- Some fields that don't indicate differences (for example, timestamps and role membership) are removed from both items.
- System managed tables, like auto aggregate, are removed.
- Items are sorted so that fields and tables appear in the same order.

Close the window when you finish examining the differences and deploy to the next stage when you're ready.

Considerations and limitations

- The *change review* feature only supports schema changes for textual item types. Currently it supports semantic models, excluding data modeling format v1, and dataflows.
- An item can be tagged as *Different*, but still not qualify to appear in the Compare window. In these cases, the **Compare** button is disabled. For example:
 - Settings changes such as name change.
 - Item type isn't yet supported.
 - Item has an unknown status because the comparison process wasn't completed.
- The content in the change review window might look a bit different than the original version since it was [modified before running the comparison](#).

Related content

[Deploy content to the next stage](#)

Last updated on 12/15/2025

Deploy content using Deployment pipelines

Any [licensed user](#) who's at least a contributor in the source stage, can deploy content to an unassigned target stage. For deploying to an existing target stage, the user must also be at least a contributor in the target stage.

You can also use the [deployment pipelines REST APIs](#) to programmatically perform deployments. For more information, see [Automate your deployment pipeline using APIs and DevOps](#).

! Note

The new Deployment pipeline user interface is currently in [preview](#). To turn on or use the new UI, see [Begin using the new UI](#).

Deploy to an empty stage

If you already have a workspace that you'd like to use with a specific stage, instead of deploying you can [assign](#) that workspace to the appropriate stage.

When you deploy content to an empty stage, the relationships between the items are kept. For example, a report that is bound to a semantic model in the source stage, is cloned alongside its semantic model, and the clones are similarly bound in the target workspace. The folder structure is also kept. If you have items in a folder in the source stage, a folder is created in the target stage. Since a folder is deployed only if one of its items is deployed, an empty folder can't be deployed.

Once the deployment is complete, refresh the semantic model. For more information, see [deploying content to an empty stage](#).

Deploying options

Deployment pipelines offer three options when it comes to deploying your Fabric content:

- [Deploy all content](#) - Deploy all your content to an adjacent stage.
- [Selective deployment](#) - Select which content to deploy to an adjacent stage.
- [Backward deployment](#) - Deploy content from a later stage to an earlier stage. Currently, this capability is available only when [deploying to an empty stage](#).

After you choose how to deploy your content, you can [Review your deployment and leave a note](#).

Deploy all content

New deploy method

1. Select the target stage.
2. From the drop-down menu, choose an adjacent stage to deploy from.
3. Select the items you want to deploy.
4. Select the **Deploy** button.

Selected stage item	Type	Compared to source	Source stage item
—	Semantic model	Only in source	Retail Analysis Sample
—	Report	Only in source	Retail Analysis Sample
—	Dashboard	Only in source	Retail Analysis Sample.pbix

The deployment process creates a duplicate workspace in the target stage. This workspace includes all the selected content from the source stage.

Selective deployment

If you don't want to deploy everything from that stage, you can select specific items for deployment. Select the **Show more** link, and then select the items you wish to deploy. When you select the **Deploy** button, only the selected items are deployed to the next stage.

Note

Items cannot be selected for deployment across workspace folders in the default stage view. However switching to [flat list view](#) allows you to select items for deployment across workspace folders.

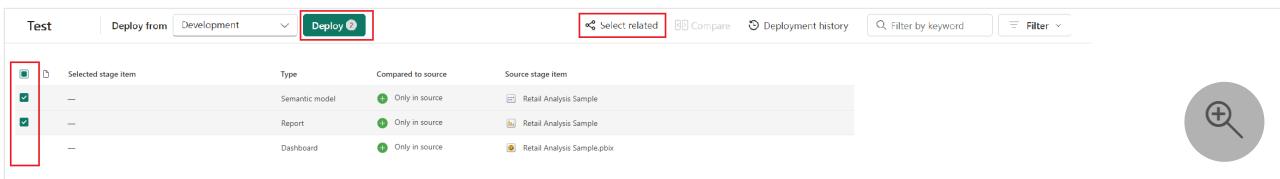
Fabric items are often related to or dependent on other items. Dashboards, reports, semantic models, dataflows, Lakehouses, and Warehouses are all examples of items that can be related to or dependent on other items. To include all items that are related to the item you want to deploy, use the select related button. For example, if you want to deploy a report to the next stage, select the **Select related** button to mark the semantic model that the report is connected to, so that both will be deployed together and the report won't break.

If you don't want to deploy everything from that stage, you can select only specific items for deployment. Since dashboards, reports, semantic models, and dataflows can have dependencies, you can use the select related button to see all the items that the selected item is dependent on. For example, if you want to deploy a report to the next stage, select the **Select related** button to mark the semantic model that the report is connected to, so that both will be deployed together and the report won't break.

New selective deploy method

The deploy button shows the number of items selected for deployment.

Unsupported items are also shown in this list. Unsupported items can't be deployed but they can be filtered.



Selected stage item	Type	Compared to source	Source stage item
Semantic model	Semantic model	Only in source	Retail Analysis Sample
Report	Report	Only in source	Retail Analysis Sample
Dashboard	Dashboard	Only in source	Retail Analysis Sample.pbix

(!) Note

- You can't deploy a Fabric item to the next stage if the items it's dependent on don't exist in the stage you are deploying to. For example, deploying a report without a semantic model will fail, unless the semantic model already exists in the target stage.
- You might get unexpected results if you choose to deploy an item without the item it's dependent on. This can happen when a semantic model or a dataflow in the target stage has changed and is no longer identical to the one in the stage you're deploying from.

When deploying workspaces that contain folders, the following rules apply:

- Items of the same name and type are paired. If there are two items with the same name and type in a workspace, then the items are paired to items in the target stage only if the path is the same (they're in the same folder).

- Since a folder is deployed only if one or more of its items is deployed, an empty folder can't be deployed.
- Individual folders can't be deployed manually in deployment. Their deployment is triggered automatically when one or more of their items is deployed.
- Deploying only some items in a folder updates the *structure* of all items in the folder in the stage being deployed to, even though the items themselves aren't deployed.
- The folder hierarchy of paired items is updated only during deployment. During assignment, after the pairing process, the hierarchy of paired items isn't updated yet.

Flat list view

With the current view of the folders hierarchy, you can select for deployment, only items in the same folder level. You cannot select items across folders.

Flat list view is an added feature of deployment pipelines that allows you to select items regardless of its location. With the flat list view, you can now select items across folders, regarding their location in the workspace.

The following are important to keep in mind when using the flat list view.

- To enable the feature there's a toggle at the top of the stage content area.
- Once in flat list view, an additional **location** column is shown and contains the full path of an item.
- The **select related** button works only in a flat list view (it's enabled when at least one item is selected). Therefore, if you are in the folders view, and click this button, then the view will automatically move to the flat list view.
- If you are in the flat list view, selected some items for deployment, and then moved back to folder view, the selection is reset to none. This behavior also applies to filtering items.

The screenshot shows the Azure Data Factory deployment pipeline interface. At the top, there are three stages: Development, Test, and Production. The Development stage has a deployment history entry for 'Demo WS [Dev]'. The Test stage also has a deployment history entry for 'Demo WS [Test]' with a timestamp of 'Deployed: 07/30/25, 5:47 PM' and a status of 'Successful deployment'. The Production stage has a deployment history entry for 'Demo WS [Prod]' with a timestamp of 'Deployed: 01/27/25, 10:26 AM' and a status of 'Successful deployment'. Below the stages, a 'Selected stage item' modal is open, listing items like 'HO', 'LH_wVarLib', 'NB - VarLib in configure', etc., with their locations and types. The 'Flat-List' button in the top right corner of the modal is highlighted with a blue box.

Location	Type	Compared to source	Source stage item
Demo WS [Dev]/Other	HomeOne (Preview)	—	—
Demo WS [Dev]/Case 1 - NB w VarLib in Configure	Lakehouse (Preview)	—	—
Demo WS [Dev]/Case 1 - NB w VarLib in Configure	SQL analytics endpoint	—	—
Demo WS [Dev]	Lakehouse (Preview)	—	—
Demo WS [Dev]	SQL analytics endpoint	—	—
Demo WS [Dev]/Case 1 - NB w VarLib in Configure	Notebook	—	—
Demo WS [Dev]/Case 3 - NB w VarLib intellisense and Ext Ite...	Variable library (Preview)	—	—
Demo WS [Dev]/Case 3 - NB w VarLib intellisense and Ext Ite...	Variable library (Preview)	—	—
Demo WS [Dev]/Case 3 - NB w VarLib intellisense and Ext Ite...	Notebook	—	—
Demo WS [Dev]/Case 2 - NB w VarLib Intellisense and same...	Notebook	—	—
Demo WS [Dev]/Other/Pipelines	Data pipeline	—	—

Backwards deployment

You might sometimes want to deploy content to a previous stage.

! Note

Be aware that backward deployment is only possible when deploying all the items. This means that you can't selectively deploy items backwards, you must deploy all the items in order to do backward deployment.

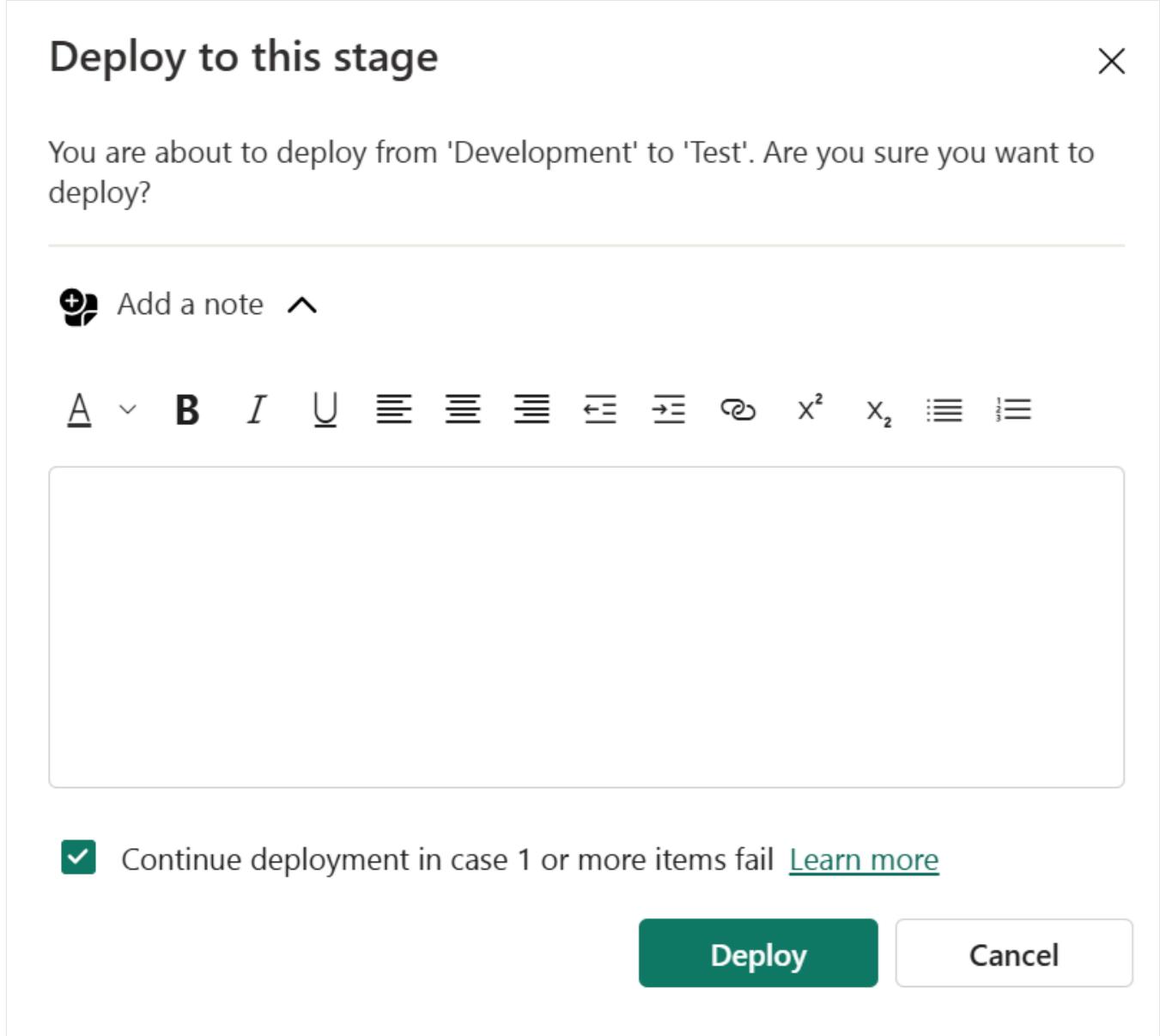
For example, if you assign an existing workspace to a production stage and then deploy it backwards, first to the test stage, and then to the development stage. Deploying to a previous stage works only if the previous stage is empty.

The screenshot shows the Azure Data Factory deployment interface. The 'Test' stage is selected. The 'Deploy from' dropdown is set to 'Production'. A modal window titled 'Selected stage item' is open, showing a list of items under the 'Production' type. The 'Type' dropdown is also set to 'Production'.

Review your deployment and leave a note

After selecting which content to deploy, a pop-up window lists all the items you're about to deploy. You can review the list and add a note, or comment, to the deployment. Adding a note is optional, but it's highly recommended as the notes are added to the [deployment history](#). With a note for each deployment, reviewing the history of your pipelines becomes more meaningful.

To leave a note, expand the **Add a note** option and write your note in the text box. When you're ready to deploy, select Deploy.



Deploy content from one stage to another

Once you have content in a pipeline stage, you can deploy it to the next stage. Deploying content to another stage is usually done after you performed some actions in the pipeline. For example, made development changes to your content in the development stage, or tested

your content in the test stage. Though you can have up to 10 different stages in the pipeline, a typical workflow for moving content is development to test stage, and then test to production. You can learn more about this process, in the [deploy content to an existing workspace](#) section.

When you deploy content to a stage that already has other content in it, select the items you want to deploy. An item that is paired with another item in the source stage (the paired item name appears on the last column) is overwritten by it.

Relationships between the items aren't kept. Therefore, if you deploy a report that is bound to a semantic model in the source stage, only the report is deployed. If you want to deploy everything connected to the report, use the **Select related** button.

To deploy content to the next stage in the deployment pipeline, select the items and then select the deploy button.

When reviewing the test and production stage cards, you can see the last deployment date and time. This indicates the last time content was deployed to the stage.

The deployment time is useful for establishing when a stage was last updated. It can also be helpful if you want to track time between test and production deployments.

Related content

- [Get started with deployment pipelines](#)
- [Deployment history](#)

Last updated on 12/15/2025

Deployment history

Deployment history is a deployment pipelines feature that enables reviewing the past deployments in your pipeline. The feature is designed as a log that lists all the past deployments in the pipeline.

You can use the deployment history log, to check the health of your pipeline, and to troubleshoot problems that occurred during previous deployments.

When you perform a deployment, you can use the built-in option to [leave notes](#), to add additional information to each deployment. Later, when you look back at your deployment history, the notes can help you understand what happened during each deployment.

Deployment history

Select a stage to view deployment history:

Deployed to	Date and time	Deployed by	Items
Production	09/19/23, 11:10 AM	 M	<div><p>All</p><p>Test</p><p>Production</p><p>All</p></div>
Test	07/25/23, 2:58 PM	 M	<div><p>✗ 1</p><p>= 2</p><p>i ✓</p></div>
Test	07/25/23, 1:28 PM	 M	<div><p>+ 3</p><p>i ✓</p></div>

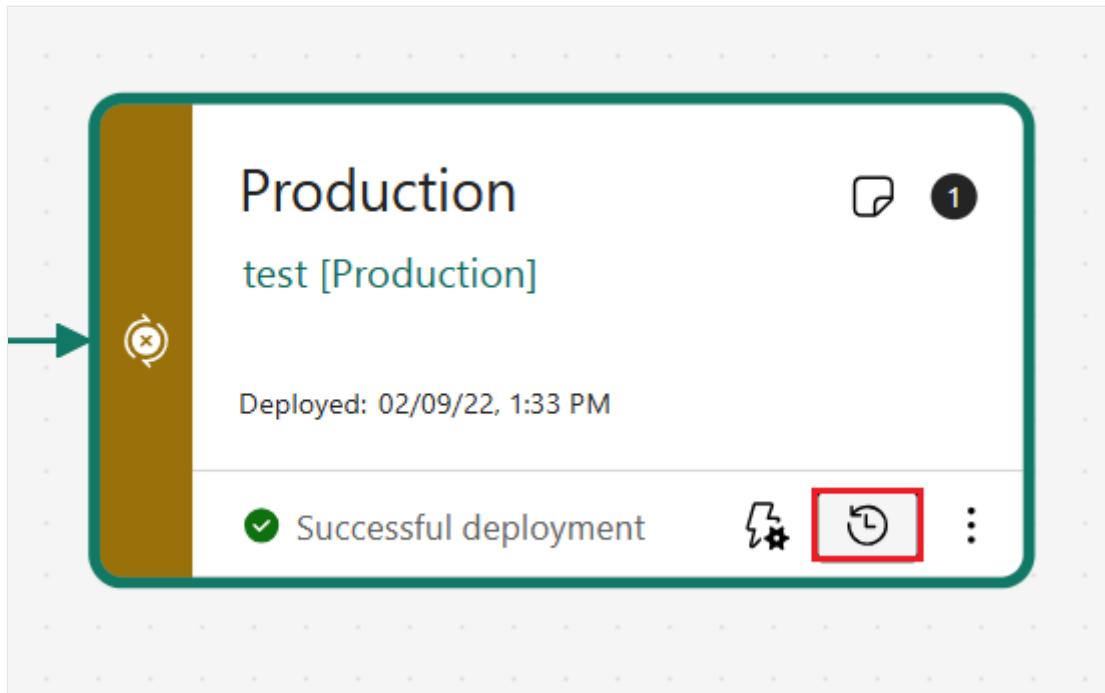
Deployment information

To view the pipeline's deployment history, select **Deployment history**.

 test-pipeline

[New Deployment pipelines](#) [Deployment history](#) [Manage Access](#) [Settings](#) ...

You can also view the deployment history of a specific stage:



The deployment history log is formatted as a table. Each row represents a single deployment, and the columns list the following information per deployment:

- **Deployed to** - The target stage of the deployment.
- **Date and time** - The date and time of the end of the deployment.
- **Deployed by** - The person (or service principal) who performed the deployment.
- **Items** - The *items* column indicates the differences between the items in the source and target stage. All labels except *failed deployment*, refer to the items in the target stage before the deployment takes place. These labels show the number of items from that category that were deployed. You can select the label to view a list of all the items in that category. If a label isn't showing, no deployed items fit that category.

[\[+\] Expand table](#)

Label	Name	Description	Expanded view
+ 3	New items	A new item that was deployed	<p>Test 07/31/22, 10:23 AM RP Rene Pelle... + 3 = 64</p> <p>New items (3)</p> <ul style="list-style-type: none"> US sales US sales Europe sales
= 6	Different items	A deployed item that's different in the source and target stages	<p>Test 07/31/22, 10:23 AM RP Rene Pelle... + 3 = 64</p> <p>Different items (6)</p> <ul style="list-style-type: none"> Global marketing Global revenue US revenue Global revenue Global revenue

Label	Name	Description	Expanded view
	Unchanged items	A deployment item that's identical to the one in the target stage	<p>Test 07/31/22, 10:23 AM RP Rene Pelle... = 64</p> <p>Unchanged items (64)</p> <ul style="list-style-type: none"> Middle east sales Australia sales Middle east marketing Afrika sales India sales
	Items failed to deploy	Indicates a failed deployment	<p>Test 06/02/22, 4:32 PM RP Rene Pelle... Failed</p> <p>Deployment was stopped Deployment couldn't be completed.</p>

- **Note** - A note, if one exists. To display the note's content, select its icon.

Test	09/11/22, 7:24 PM	RP	
Test deployment 09/11/22, 7:24 PM			
Here's an example of a note you can add when you deploy content.			

- **ID** - The deployment ID. Use as a reference when troubleshooting your latest deployment if it fails.
- **Status** - The status of the deployment.

Expand table

Icon	Deployment status
	Successful
	Unsuccessful

You can choose to see the deployment history of a specific stage, or of all stages.

Considerations and limitations

The following section lists the deployment history limitations.

- Detailed information is displayed from June 2022. Before this date, only summarized information is displayed.
- The deployment history log lists up to 1000 most recent deployments.

- Deployment history displays the name of the item during deployment. When you change an item's name, deployment history treats it as a new item. In that case, the item with the changed name appears in the log for the first time, after it's deployed.

Related content

- [Get started with deployment pipelines](#)
- [Create deployment rules](#)

Last updated on 12/15/2025

Create deployment rules

When you're working in a deployment pipeline, different stages might have different configurations. For example, each stage can have different databases or different query parameters. The development stage might query sample data from the database, while the test and production stages query the entire database.

When you deploy content between pipeline stages, you can configure deployment rules to change the content while keeping some settings intact. For example, you can define a rule for semantic model in a production stage to point to a production database instead of one in the test stage. The rule is defined in the production stage, under the appropriate semantic model. Once the rule is defined, content deployed from test to production inherits the value as defined in the deployment rule. This rule always applies as long as it's unchanged and valid.

! Note

The new Deployment pipeline user interface is currently in [preview](#). To turn on or use the new UI, see [Begin using the new UI](#).

You can configure data source rules, parameter rules, and default lakehouse rules. The following table lists the type of items you can configure rules for, and the type of rule you can configure for each one.

[] [Expand table](#)

Item	Data source rule	Parameter rule	Default lakehouse rule	Details
Dataflow gen1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Use to determine the values of the data sources or parameters for a specific dataflow gen1.
Semantic model	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Use to determine the values of the data sources or parameters for a specific semantic model.
Datamart	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Use to determine the values of the data sources or parameters for a specific datamart.
Paginated report	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Defined for the data sources of each paginated report. Use to determine the data sources of the paginated report.

Item	Data source rule	Parameter rule	Default lakehouse rule	Details
Mirrored database	✓	✗	✗	Defined for the data sources of each mirrored database.
Notebook	✗	✗	✓	Use to determine the default lakehouse for a specific notebook.

! Note

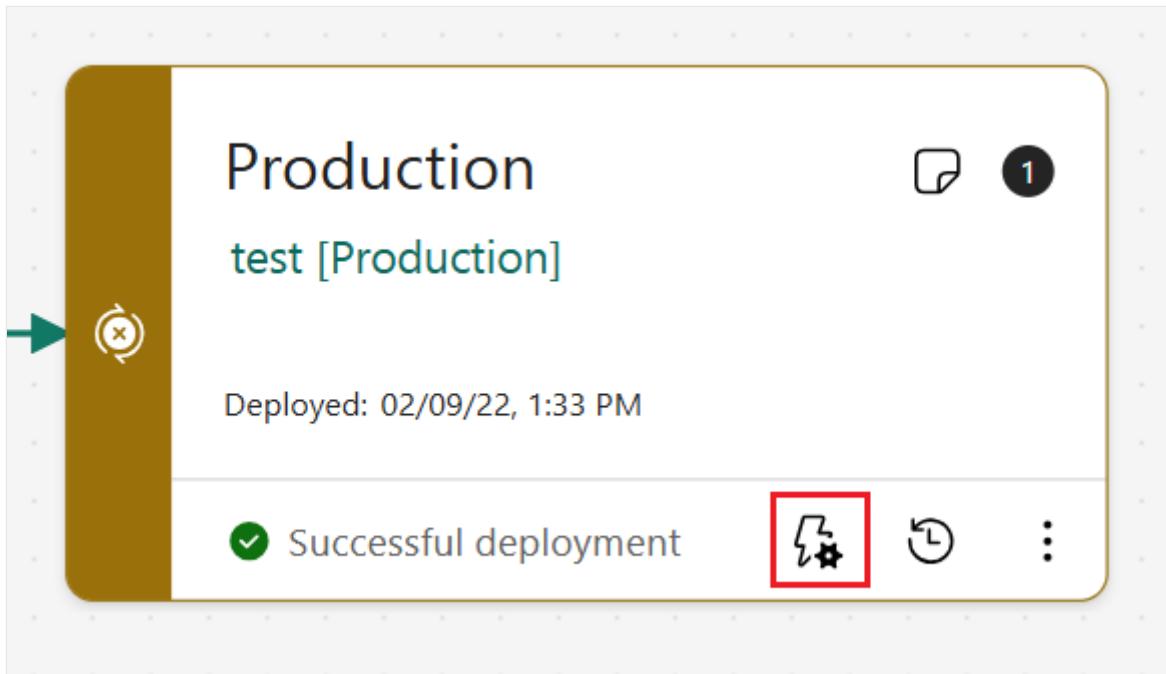
Data source rules only work when you change data sources from the same type.

Create a deployment rule

To create a deployment rule, follow the steps in this section. After you create all the deployment rules you need, deploy the semantic models with the newly created rules from the source stage to the target stage where the rules were created. Your rules don't apply until you deploy the semantic models from the source to the target stage.

Create a deployment rule in the new UI

1. In the pipeline stage you want to create a deployment rule for, select **Deployment rules**.



2. A list of items you can set rules for appear in the window. Not all items in the pipeline are listed. Only items of a type that you can create rules for are listed (dataflows gen1, semantic model, datamarts, notebooks, and paginated reports). To find the item you want to set a rule for, use the search or filter functionalities.

Deployment rules [Production]

Use rules to define how your content is deployed to this stage. [Learn more](#)

Name
V3-SqlAzure-Cached
V3-SqlAzure-DQ

Filter by keyword Filter (1)

Type (1)

Dataflow

Semantic model

3. Select the item you want to create a rule for. The types of rules you can create for that item are displayed. So, for example, if you're creating a rule for a dataflow gen1, you can create a data source rule or a parameter rule. If you're creating a rule for a notebook, you can create a default lakehouse rule.

4. Select the type of rule you want to create, expand the list, and then select **Add rule**. There are two types of rules you can create:

Semantic model deployment rules

Set deployment rules

Create rules to run when deploying this item. [Learn more ↗](#)

⚡ Data source rules

^

Define the data source that will be connected to this item. [Learn more](#)



No rules were added yet

+ Add rule

⚡ Parameter rules

^

Define the parameter values to apply to this item. [Learn more](#)



No rules were added yet

+ Add rule

- **Data source rules**

From the data source list, select a data source name to be updated. Use one of the following methods to select a value to replace the one from the source stage:

- Select from a list.

- Select *Other* and manually add the new data source. You can only change to a data source from the same type.

! Note

- *Data source rules* will be grayed out if you're not the owner of the item you're creating a rule for, or if your item doesn't contain any data sources.
- For *dataflows gen1*, *semantic models* and *paginated reports*, the data source list is taken from the source pipeline stage.
- You can't use the same data source in more than one rule.

- **Parameter rules** Select a parameter from the list of parameters; the current value is shown. Edit the value to the value you want to take effect after each deployment.
- **Default lakehouse rules** This rule only applies to notebooks. Select a lakehouse to connect to the notebook in the target stage and set it as its default. For more information, see [Notebook in deployment pipelines](#).

Supported data sources for dataflow gen1 and semantic model rules

Data source rules can be defined for the following data sources:

- Azure Analysis Services (AAS)
- Azure Synapse
- SQL Server Analysis Services (SSAS)
- Azure SQL Server
- SQL server
- Odata Feed
- Oracle
- SapHana (import mode only; not direct query mode)
- SharePoint
- Teradata

For other data sources, we recommend [using parameters to configure your data source](#).

Considerations and limitations

This section lists the limitations for the deployment rules.

- To create a deployment rule, you must be the owner of the item you're creating a rule for.
- Deployment rules can't be created in the development stage.
- When an item is removed or deleted, its rules are deleted too. These rules can't be restored.
- When you unassign and reassign a workspace to [reestablish connections](#), rules for that workspace are lost. To use these rules again, reconfigure them.
- If the data source or parameter defined in a rule is changed or removed from the item it points to in the source stage, the rule isn't valid anymore, and deployment fails.
- After you deploy a paginated report with a data source rule, you can't open the report using [Power BI Report Builder](#).
- Deployment rules only take effect the next time you deploy to that stage. However, if you create rules and then compare the stages before you deploy, the comparison is done based on the rules that were created even though they didn't take effect yet.
- The following scenarios aren't supported:
 - Data source rules for dataflows gen1 that have other dataflows as sources.
 - Data source rules for common data model (CDM) folders in a dataflow gen1.
 - Data source rules for semantic models that use dataflows gen1 as their source.
 - Creating data source rules on a semantic model that uses Native query and DirectQuery together.
 - Parameter rules aren't supported for paginated reports.
 - Adding data source rules for semantic models and dataflows gen1 on data sources which are parametrized.

Related content

- [Get started with deployment pipelines](#)
- [Automate your deployment pipeline using APIs and DevOps](#)

Automate your deployment pipeline for Power BI items by using APIs

The Microsoft Power BI [deployment pipelines](#) tool enables business intelligence teams to build an efficient and reusable release process for their Power BI content.

Note

The deployment pipelines APIs listed here only apply to Power BI items. For Fabric APIs, see the [Fabric API documentation](#).

To achieve continuous integration and continuous delivery (CI/CD) of content, many organizations use automation tools, including [Azure DevOps](#). Organizations that use Azure DevOps, can use the [Power BI automation tools](#) extension, which supports many of the deployment pipelines API operations.

You can use the [deployment pipelines Power BI REST APIs](#) to integrate Fabric into your organization's automation process. Here are a few examples of what can be done by using the APIs:

- Manage pipelines from start to finish, including creating a pipeline, assigning a workspace to any stage, and deploying and deleting the pipeline.
- Assign and unassign users to and from a pipeline.
- Integrate Fabric into familiar DevOps tools such as [Azure DevOps](#) or [GitHub Actions](#) ↗.
- Schedule pipeline deployments to happen automatically at a specific time.
- Deploy multiple pipelines at the same time.
- Cascade depending on pipeline deployments. If you have content connected across pipelines, you can make sure some pipelines are deployed before others.

Prerequisites

Before you use the deployment pipelines APIs, make sure of the following:

- The *user or service principal* that calls the APIs has [pipeline and workspace permissions](#) and access to an [Microsoft Entra application](#).
- If you're using PowerShell scripts, install the Power BI PowerShell cmdlets [Install-Module MicrosoftPowerBIMgmt](#).

Deployment pipelines API functions

The [deployment pipelines Power BI REST APIs](#) allow you to perform the following functions:

- **Get pipeline information** - Retrieve information about your pipelines and their content. Getting the pipeline information enables you to dynamically build the deployment API calls. You can also check the [status of a deployment](#) or the [deployment history](#).
- **Deploy** - The REST calls enable developers to use any type of deployment available in the Fabric service.
- **Create and delete pipelines** - Use [Create pipeline](#) and [Delete pipeline](#) to perform these operations.
- **Manage workspaces** - With [Assign workspace](#) and [Unassign workspace](#), you can assign and unassign workspaces to specific pipeline stages.
- **Manage pipeline users** - [Delete pipeline user](#) lets you remove a user from a pipeline. [Update pipeline user](#) allows you to add a user to your pipeline.

Which deployment types do the APIs support?

The APIs support the following deployment types:

- **Deploy all** - A single API call that deploys all the content in the workspace to the next stage in the pipeline. For this operation, use the [Deploy all](#) API.
- **Selective deploy** - Deploys only specific items, such as reports or dashboards, in the pipeline. For this operation, use the [Selective deploy](#) API.
- **Backward deploy** - Deploys new items to the previous stage. Backward deployment only works if the items that are deployed don't already exist in the target stage. For this operation, use either the [Deploy all](#) or the [Selective deploy](#) APIs, with `isBackwardDeployment` set to `True`.
- **Update App** - As part of the deployment API call, you can update the content of the app related to that stage. Updated items are automatically available to your end users, after a deployment is complete. For this operation, use either the [Deploy all](#) or the [Selective deploy](#) APIs, with `PipelineUpdateAppSettings`.

Integrate your pipeline with Azure DevOps

To automate the deployment processes from within your [release pipeline in Azure DevOps](#), use one of these methods:

- **PowerShell** - The script signs into Fabric using a *service principal* or a *user*.
- **Power BI automation tools** - This extension works with a *service principal* or a *user*.

You can also use other [Power BI REST API](#) calls, to complete related operations such as importing a *.pbix* into the pipeline, updating data sources and parameters.

Use the Power BI automation tools extension

The Power BI automation tools extension is an [open source](#)  Azure DevOps extension that provides a range of deployment pipelines operations that can be performed in Azure DevOps. The extension eliminates the need for APIs or scripts to manage pipelines. Each operation can be used individually to perform a task, such as creating a pipeline. Operations can be used together in an Azure DevOps pipeline to create a more complex scenario, such as creating a pipeline, assigning a workspace to the pipeline, adding users, and deploying.

After you add the [Power BI automation tools](#)  extension to DevOps, you need to create a service connection. The following connections are available:

- **Service principal** (recommended) - This connection authenticates by using a [service principal](#) and requires the Microsoft Entra app's secret and application ID. When you use this option, verify that the [service admin settings](#) for the service principal are enabled.
- **Username and password** – Configured as a generic service connection with a username and a password. This connection method doesn't support multifactor authentication. We recommend that you use the service principal connection method because it doesn't require storing user credentials on Azure DevOps.

Note

The Power BI automation tools extension uses an Azure DevOps service connection to store credentials. For more information, see [How we store your credentials for Azure DevOps Services](#).

After you enable a service connection for your Azure DevOps Power BI automation tools, you can [create pipeline tasks](#). The extension includes the following deployment pipelines tasks:

- Create a new pipeline
- Assign a workspace to a pipeline stage

- Add a user to a deployment pipeline
- Add a user to a workspace
- Deploy content to a deployment pipeline
- Remove a workspace from a deployment pipeline
- Delete a pipeline

Access the PowerShell samples

You can use the following PowerShell scripts to understand how to perform several automation processes. To view or copy the text in a PowerShell sample, use the links in this section.

You can also download the entire [PowerBI-Developer-Samples](#) GitHub folder.

- [Deploy all ↗](#)
- [Selective deployment ↗](#)
- [Wait for deployment ↗](#)
- [End to end example of pipeline creation and backward deployment ↗](#)
- [Assign an admin user to a pipeline ↗](#)

PowerShell example

This section describes an example PowerShell script that deploys a semantic model, report, and dashboard, from the development stage to the test stage. The script then checks whether the deployment was successful.

To run a PowerShell script that performs a deployment, you need the following components.

You can add any of these parts into [tasks](#) in your Azure pipeline stages.

- 1. Sign in** - Before you can deploy your content, you need to sign in to Fabric using a *service principal* or a *user*. Use the [Connect-PowerBIServiceAccount](#) command to sign in.
- 2. Build your request body** - In this part of the script you specify which items (such as reports and dashboards) you're deploying.

PowerShell

```
$body = @{
    sourceStageOrder = 0 # The order of the source stage. Development (0), Test
```

```

(1).
datasets = @(
    @{sourceId = "Insert your dataset ID here" }
)
reports = @(
    @{sourceId = "Insert your report ID here" }
)
dashboards = @(
    @{sourceId = "Insert your dashboard ID here" }
)

options = @{
    # Allows creating new item if needed on the Test stage workspace
    allowCreateArtifact = $TRUE

    # Allows overwriting existing item if needed on the Test stage
    workspace
    allowOverwriteArtifact = $TRUE
}
} | ConvertTo-Json

```

3. Deploy - Here you perform the deployment.

PowerShell

```

$url = "pipelines/{0}/Deploy" -f "Insert you pipeline ID here"
$deployResult = Invoke-PowerBIRestMethod -Url $url -Method Post -Body $body |
ConvertFrom-Json

```

4. (Optional) Deployment completion notification - Because the deployment API is asynchronous, you can program the script to notify you when the deployment is complete.

PowerShell

```

$url = "pipelines/{0}/Operations/{1}" -f "Insert you pipeline ID
here",$deployResult.id
$operation = Invoke-PowerBIRestMethod -Url $url -Method Get | ConvertFrom-Json
while($operation.Status -eq "NotStarted" -or $operation.Status -eq "Executing")
{
    # Sleep for 5 seconds
    Start-Sleep -s 5
    $operation = Invoke-PowerBIRestMethod -Url $url -Method Get | ConvertFrom-
Json
}

```

Considerations and limitations

- Deployment by using APIs is subject to the same [limitations](#) as the deployment pipelines user interface.
- A *service principal* can't configure *OAuth* credentials. After you deploy new items, the signed in *service principal* becomes the owner of any deployed paginated reports and semantic models. In such cases, a refresh can't be completed.
- Deploying dataflows by using a *service principal* isn't supported.
- The maximum number of items that can be deployed in a single deployment is 300.
- The deployment pipelines APIs currently only support Power BI items.
- Creating a customized pipeline of 2-10 stages is currently supported only through the UI.

Related content

- [Get started with deployment pipelines](#)
- [Deployment pipelines best practices](#)
- [Troubleshooting deployment pipelines](#)

Last updated on 12/15/2025

Automate your deployment pipeline with Fabric APIs

The Microsoft Fabric [deployment pipelines](#) tool enables teams to build an efficient and reusable release process for their Fabric content.

Use the [deployment pipelines Fabric REST APIs](#) to integrate Fabric into your organization's automation process. Here are a few examples of what can be done by using the APIs:

- Integrate Fabric into familiar DevOps tools such as Azure DevOps or GitHub Actions.
- Schedule pipeline deployments to happen automatically at a specific time.
- Deploy multiple pipelines at the same time.
- Cascade depending on pipeline deployments. If you have content connected across pipelines, you can make sure some pipelines are deployed before others.

Prerequisites

To work with deployment pipeline APIs, you need the following prerequisites:

- The same [prerequisites you need to use deployment pipelines](#).
- A Microsoft Entra token for Fabric service. Use that token in the authorization header of the API call. For information about how to get a token, see [Fabric API quickstart](#).

You can use the REST APIs without PowerShell, but the scripts in this article use PowerShell. To run the scripts, you need to install the following programs:

- [PowerShell](#)
- [Azure PowerShell Az module](#)

Deployment pipelines API functions

The [deployment pipelines Fabric REST APIs](#) allow you to perform the following functions:

- [Get Deployment Pipeline](#): Returns information about the specified deployment pipeline.
- [List Deployment Pipelines](#): Returns a list of deployment pipelines that the user has access to.
- [List Deployment Pipeline Stages](#): Returns the stages of the specified deployment, including its ID, display name, description, and whether the stage is public or not.

- [List Deployment Pipeline Stage Items](#): Returns the supported items from the workspace assigned to the specified stage of the specified deployment pipeline.
- [Deploy Stage Content](#): Deploys items from the specified stage of the specified deployment pipeline.
 - Use this API to deploy all items or to select specific items to deploy. If no specific items are selected, all items are deployed.
 - To find the relevant stage ID to deploy, use the [List Deployment Pipeline Stages](#) API.
 - This API is integrated with the [Long Running Operations APIs](#) to monitor the deployment status.
 - Get the operation state to see if the operation is complete with the [Get Long Running - Get Operation state](#) API.
 - For 24 hours after the deployment is completed, the extended deployment information is available in the[Get Operation Result](#) API.
- [Create deployment pipeline](#): Create a Deployment Pipeline.
- [Delete deployment pipeline](#): Delete a Deployment Pipeline.
- [Update deployment pipeline](#): Update a Deployment Pipeline.
- [Get deployment pipeline stage](#): Get details of a Deployment Pipeline Stage.
- [Update deployment pipeline stage](#): Update a Deployment Pipeline Stage.
- [Add deployment pipeline role assignment](#): Add a role assignment to a deployment pipeline.
- [Delete deployment pipeline role assignment](#): Delete a role assignment from a deployment pipeline.
- [List deployment pipeline role assignments](#): List all role assignments for a deployment pipeline.
- [Assign workspace to deployment pipeline stage](#): Assign a workspace to a specific deployment pipeline stage.
- [Unassign workspace from deployment pipeline stage](#): Unassign a workspace from a specific deployment pipeline stage.
- [Get deployment pipeline operation](#): Get details of a deployment pipeline operation.
- [List deployment pipeline operations](#): List all operations for a deployment pipeline.

You can also use other [Fabric REST API](#) calls, to complete related operations.

PowerShell examples

You can use the following PowerShell scripts to understand how to perform several automation processes. To view or copy the text in a PowerShell sample, use the links in this section.

You can also download the entire [Fabric-Samples](#) GitHub folder.

- [Deploy all](#)

Provide the following information:

- Pipeline name
- Source stage name
- Target stage name
- Deployment notes (optional)
- Principal type. Choose either *UserPrincipal* or *ServicePrincipal*. If service principal, also provide:
 - Application (client) ID of the service principal
 - Directory (tenant) ID of the service principal
 - Secret value of the service principal

- [Selective deploy](#)

Provide the following information:

- Pipeline name
- Source stage name
- Target stage name
- Items to deploy (items display name and item type)
- Deployment notes (optional)
- Principal type. Choose either *UserPrincipal* or *ServicePrincipal*. If service principal, also provide:
 - Application (client) ID of the service principal
 - Directory (tenant) ID of the service principal
 - Secret value of the service principal

- [Assign to new deployment pipeline and deploy](#)

Provide the following information:

- Development workspace ID
- New production workspace name
- Pipeline name
- Deployment notes (optional)

- Principal type. Choose either *UserPrincipal* or *ServicePrincipal*. If service principal, also provide:
 - Application (client) ID of the service principal
 - Directory (tenant) ID of the service principal
 - Secret value of the service principal

Considerations and limitations

When using the deployment pipelines APIs, consider the following limitations:

- All limitations that apply for deployment pipeline, apply when using the APIs. For more information, see [Deployment pipelines best practices](#).
- *Dataflows* are currently not supported. Customers using dataflows can use the [Power BI APIs](#).
- Not all deployment options available in the Power BI APIs are available in Fabric. The following APIs *aren't* available in Fabric's Deploy stage content API:
 - allowPurgeData
 - allowTakeOver
 - allowSkipTilesWithMissingPrerequisites

To use one of these APIs, use the [Power BI API](#) to deploy. However, these APIs only work for Power BI items.

Related content

- [Get started with deployment pipelines](#)
- [Deployment pipelines best practices](#)
- [Troubleshooting deployment pipelines](#)

Last updated on 12/15/2025

What is a variable library?

A Microsoft Fabric variable library is a bucket of variables that other items in the workspace can consume as part of application lifecycle management (ALM). It functions as an item within the workspace that contains a list of variables, along with their respective values for each stage of the release pipeline. It presents a unified approach for efficient management of item configurations within a workspace, to help ensure scalability and consistency across lifecycle stages.

For example, a variable library can contain variables that hold values for:

- An integer to be used in a wait activity in a pipeline.
- A lakehouse reference to be the source in a *copy data* activity. Each value is used in a different pipeline, based on the release stage of the pipeline.
- A lakehouse reference to be configured as a notebook default lakehouse. Each value is used in a different pipeline, based on the release stage of the notebook.

Value resolution in the consumer item isn't necessarily tied to its deployment. Rather, each consumer item resolves the value based on its own context.

The experience of a variable library differs based on the variable type, but all variable libraries allow you to define and manage variables that other items can use.

A Fabric variable library:

- Is compatible with continuous integration and continuous delivery (CI/CD) processes. This compatibility allows [integration with Git](#) and deployment through [deployment pipelines](#).
- Supports automation via Fabric public APIs.

Benefits

Variable libraries enable customers to customize and share configurations.

Customize configurations

You can configure a variable value based on the release pipeline stage. You can configure the variable library with sets of values: one value for each stage of the release pipeline. Then, after one-time settings of the active value set for each stage, the correct value is automatically used in the pipeline stage. Examples include:

- Changing an item's connection based on the stage.
- Switching to a different cloud data source based on the stage.
- Adjusting data quantity in a query based on the stage.

Share configurations

Variable libraries provide a centralized way to manage configurations across the workspace items. For example, if you have several lakehouses in the workspace and each one has a shortcut that uses the same data source, you can create a variable library with that data source as one of the variables. That way, if you want to change the data source, you have to change it only once in the variable library. You don't need to change it in each lakehouse separately.

Variable library structure

Variable libraries contain one or more variables. Each variable has a name, type, and default value. You can also add a note to each variable to describe its purpose or how to use it.

The screenshot shows the 'Prediction model variables' section of a variable library. It lists five variables:

Name *	Type	Default value set *
StudiedHoursForPredictedScore	Integer	6
Model training dataset	String	{'Hours': [2, 3, 4, 5, 6, 7, 8, 9, 10]}
NYC pipeline source	String	S:\\Data\\Clean\\CountryScoresDat...
Refresh data	Boolean	True
Report datasource	String	'Test scores DB'/NYC_Test

Default value

The default value is the value that's used unless you specifically define a different value.

All variables must have a default value. If the variable type is *string*, the default value can be `null`.

Alternative value sets

Value sets define the values of each variable in the variable library. A variable library typically contains multiple value sets. The active (or effective) value set contains the value that the

consumer item receives for that workspace.

In each workspace, you select a value set to be active. The active value set of a workspace doesn't change during a deployment or update from Git.

The screenshot shows the 'Prediction model variables' interface. On the left, a table lists variables with columns for Name, Note, Value type, and Default value set. The 'Default value set' column contains pointers to the 'Test value set' on the right. The 'Test value set' contains five entries: '2', '['Hours': [2, 3, 4, 5, 6, 7, 8, 9, 10]]', 'S:\\Data\\Clean\\CountryScoresDat...', 'True', and "'Test scores DB'/NYC_Test2'".

Prediction model variables					
Variables			Alternative value sets		
Name *	Note	Value type	Default value set *	...	Test value set *
StudiedHoursForPredictedScore		Integer	6	...	2
Model training dataset		String	{'Hours': [2, 3, 4, 5, 6, 7, 8, 9, 10]}	...	['Hours': [2, 3, 4, 5, 6, 7, 8, 9, 10]]
NYC pipeline source		String	S:\\Data\\Clean\\CountryScoresDat...	...	S:\\Data\\Clean\\CountryScoresDat...
Refresh data		Boolean	True	...	True
Report datasource		String	'Test scores DB'/NYC_Test	...	'Test scores DB'/NYC_Test2

When you create an alternative value set, the new value set is created with pointers to the default value for each variable. You can then change the value for each variable in the new value set.

Supported items

The following items support the variable library:

- [Pipeline](#)
- [Shortcut for a lakehouse](#)
- Notebook, through [NotebookUtils](#) and [%%configure](#)
- [Dataflow Gen 2](#)
- [Copy job](#)
- [User data functions](#)

Considerations and limitations

Size limitations

- There can be *up to 1,000 variables* and *up to 1,000 value sets*, as long as you meet both of these requirements:
 - The total number of cells in the alternative value sets is less than 10,000.
 - The item's size doesn't exceed 1 MB.

These requirements are validated when you save changes.

- The note field can have up to 2,048 characters.
- The value-set description field can have up to 2,048 characters.

Limitations for alternative value sets

- Alternative value sets in a variable library appear in the order in which you added them. Currently, you can't reorder them in the UI. To change the order, edit the JSON file directly.
- The name of each value set must be unique within a variable library.
- Variable names must be unique within a variable library. You can have two variables with the same name in a workspace if they're in different items.
- There's always one (and only one) active value set in a variable library at a time. You can't delete a value set while it's active. To delete it, first configure another value set to be active. You can have a different active value set for each stage of a deployment pipeline.

Related content

- [Variable library permissions](#)
- [Create and manage variable libraries](#)

Last updated on 12/15/2025

Create and manage variable libraries

Microsoft Fabric variable libraries enable developers to customize and share item configurations within a workspace, with a goal of streamlining content lifecycle management. This article explains how to create, manage, and consume variable libraries.

For a more detailed walkthrough of the process, see the [tutorial for using variable libraries](#).

Prerequisites

To create variable library items in Fabric, you need:

- A Fabric tenant account with an active subscription. [Create an account for free](#).
- A [workspace](#) with a Microsoft Fabric-enabled [capacity](#).
- The following [tenant switches](#) enabled from the Admin portal:
 - [Users can create Fabric items](#)

The tenant admin, capacity admin, or workspace admin can enable these switches, depending on your [organization's settings](#).

Security considerations and permissions management for Fabric variable libraries

Fabric variable libraries are powerful constructs that enable centralized management of variables across multiple Fabric items. However, this flexibility introduces critical security considerations.

Because variable libraries themselves are Fabric items, they're governed by their own [permission](#) sets. These permission sets might differ from those of the items that consume their variables. This discrepancy can lead to scenarios where a user has write access to a variable library but lacks any access to the consuming item.

In such cases, unauthorized users can modify variable values in ways that intentionally or unintentionally alter the behavior of dependent Fabric items. This ability creates a potential attack vector where malicious updates to shared variables could compromise the integrity, security, or functionality of those items.

To mitigate these risks, follow these key practices:

- **Adopt strict permission controls:** Administrators must carefully manage write [permissions](#) on variable libraries so that only trusted users or services can modify them.

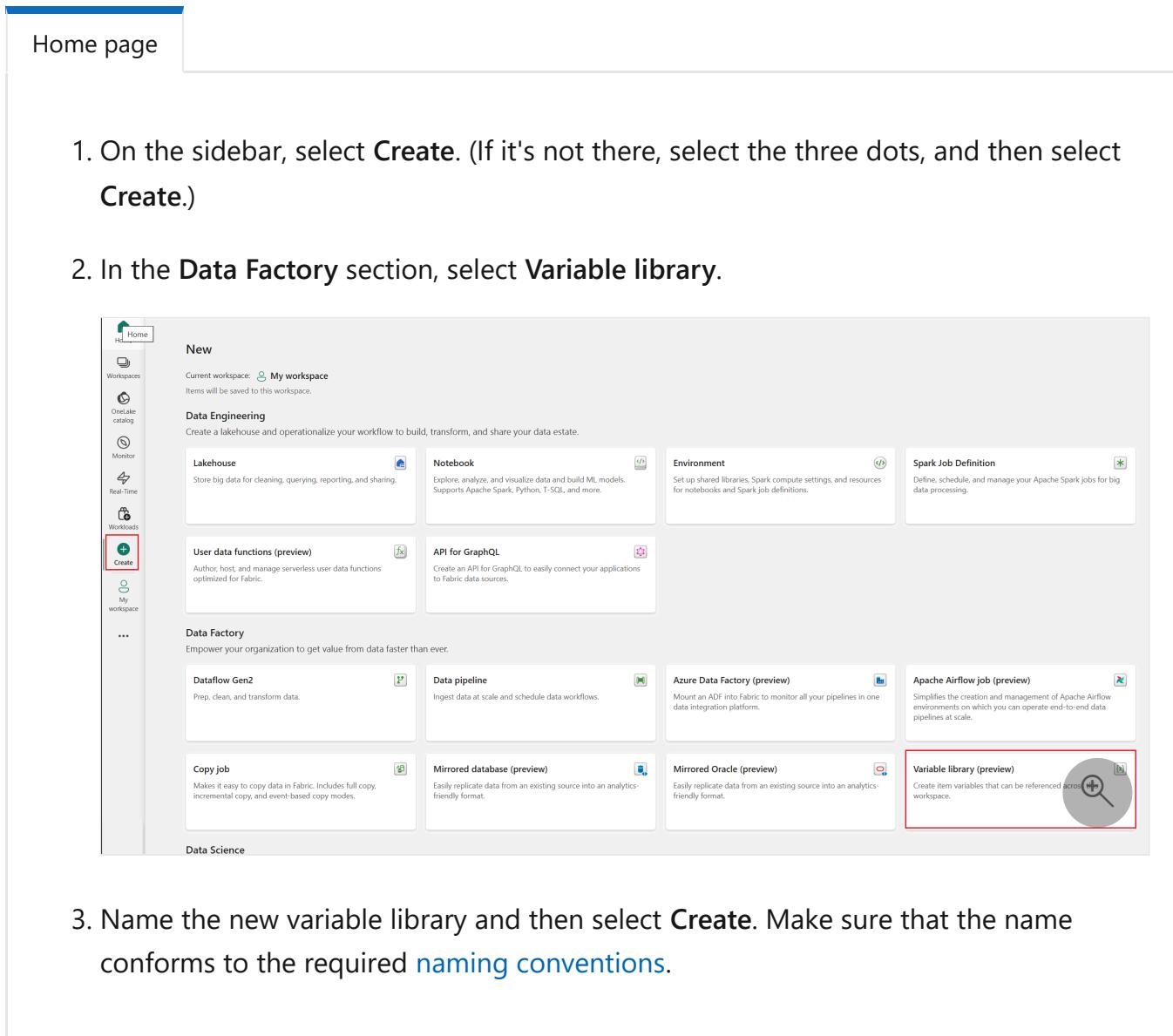
This practice includes avoiding overly permissive access and regularly auditing permission assignments.

- **Use trusted library references:** Items should reference variables only from libraries that are explicitly designated as trusted. This trust model should be enforced through governance policies that validate the source of variable references during development and deployment.

For more information, see [Variable library permissions](#).

Create a variable library item

You can create a variable library item from the Fabric home page or from inside your workspace:



The screenshot shows the Fabric Home page. On the left, there's a sidebar with various icons: Home, Workspaces, OneLake catalog, Monitor, Real-Time, Workloads, and a red-highlighted 'Create' button. Below the sidebar, the main area has sections for Data Engineering, Data Factory, and Data Science. In the Data Engineering section, the 'Variable library (preview)' option is highlighted with a red box. The 'Create' button in the sidebar is also highlighted with a red box.

1. On the sidebar, select **Create**. (If it's not there, select the three dots, and then select **Create**.)
2. In the **Data Factory** section, select **Variable library**.

3. Name the new variable library and then select **Create**. Make sure that the name conforms to the required [naming conventions](#).

An empty variable library appears. You can now add variables to it.

Home

Save + New variable Delete variable Add value set

EmployeeInfoVL

There are no variables

All variables you choose to create will appear here. [Learn more](#)

New variable

Manage variable libraries and their variables

You can manage the variables in the variable library from the top menu bar.

Home

Save + New variable Delete variable Add value set

Prediction model variables

Variables	Name *	Note	Value type	Default value set *	...
<input type="checkbox"/>	StudiedHoursForPredictedScore		Integer	6	
<input type="checkbox"/>	Model training dataset		String	{'Hours': [2, 3, 4, 5, 6, 7, 8, 9, 10]}	
<input type="checkbox"/>	NYC pipeline source		String	S:\\Data\\Clean\\CountryScoresDat...	
<input type="checkbox"/>	Refresh data		Boolean	True	

Add a variable

To add a new variable to the library:

1. Select **+ New variable**.
2. Enter a name. Make sure that it follows the [naming conventions](#).
3. In the dropdown list, select a type. [See a list of supported variable types](#).

4. Enter a default value.
5. Add a note that explains what the variable is for or how to use it (optional).
6. Select **Save**.

Delete or edit a variable

- To delete a variable, select one or more variables and then select **Delete variable > Save**.
- To edit the name, type, or value set of a variable, change the value and then select **Save**.
- To add another alternative value set, select **Add value set**.

Note

Selecting **Save** after editing any variable in the variable library triggers an error validation check to make sure that all the variable names and values are valid. You must fix any errors before you save the changes.

Add a value set

To add another value set that you can use in a different stage:

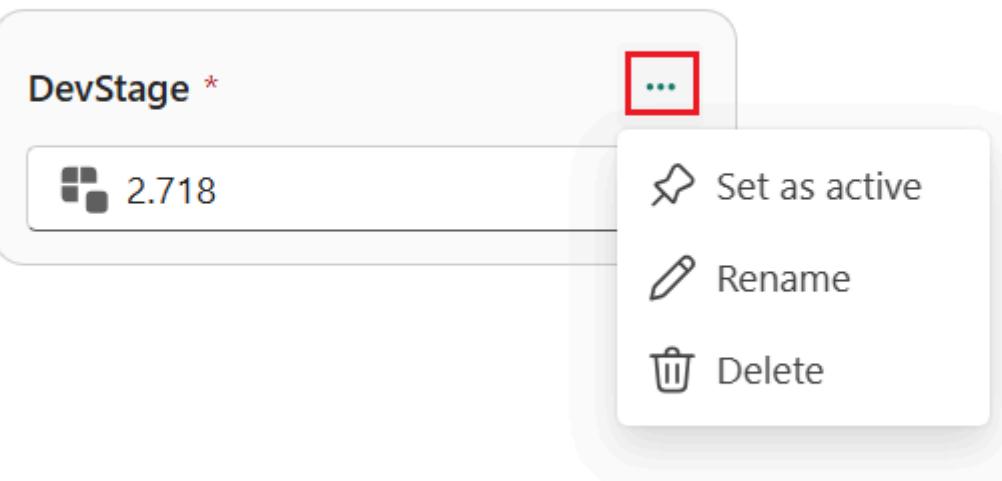
1. Select **Add value set**.
2. Name the value set. Make sure that it follows the [naming conventions](#). Give it a description (optional) of up to 2,048 characters.
3. If you want to use this value to be the currently active value set in this workspace, select **Set as active**.
4. Enter values for all the variables in the variable library.
5. Select **Save**.

Edit a value set

To edit a value set:

1. Select the three dots next to the name of the value set.
2. Select **Set as active** (for this workspace), **Rename**, or **Delete**.

Alternative value sets



3. Select **Save**. Changes take effect only after you save them.

To reset the value of each variable to the default value, select the reset button.

The screenshot shows the 'VS2' value set with three variables:

- Variable 1: Value 123
- Variable 2: Value 20
- Variable 3: Value 2025-06-14T16:15:20.123Z

Considerations and limitations

Size limitations

- There can be *up to 1,000 variables* and *up to 1,000 value sets*, as long as you meet both of these requirements:
 - The total number of cells in the alternative value sets is less than 10,000.
 - The item's size doesn't exceed 1 MB.

These requirements are validated when you save changes.

- The note field can have up to 2,048 characters.
- The value-set description field can have up to 2,048 characters.

Limitations for alternative value sets

- Alternative value sets in a variable library appear in the order in which you added them. Currently, you can't reorder them in the UI. To change the order, edit the JSON file directly.
- The name of each value set must be unique within a variable library.
- Variable names must be unique within a variable library. You can have two variables with the same name in a workspace if they're in different items.
- There's always one (and only one) active value set in a variable library at a time. You can't delete a value set while it's active. To delete it, first configure another value set to be active. You can have a different active value set for each stage of a deployment pipeline.

Last updated on 12/15/2025

Variable names and types

A variable library item in Microsoft Fabric contains a list of variables and their default values. It can also contain other value sets that hold alternative values.

Each variable in the variable library has the following properties:

- Name
- Note (optional), up to 2,048 characters
- Type
- Default value set
- Alternative value sets (optional)

Naming conventions

Variable library name

The name of variable library item itself must follow these conventions:

- Isn't empty
- Doesn't have leading or trailing spaces
- Starts with a letter
- Can include letters, numbers, underscores, hyphens, and spaces
- Doesn't exceed 256 characters in length

The variable library name is *not* case sensitive.

Variable name

The name of a variable inside the variable library must follow these conventions:

- Isn't empty
- Doesn't have leading or trailing spaces
- Starts with a letter or an underscore
- Can include letters, numbers, underscores, and hyphens
- Doesn't exceed 256 characters in length

The variable name is *not* case sensitive.

Value set name

Value set names have the same restrictions as [variable names](#).

Variable types

Before you can add a value to a variable, you must define the variable type. The variables in the variable library can be any of the following types:

- **String:** Any character. Can be `null` or empty.
- **Boolean:** `True` or `False`.
- **DateTime:** Date and time represented as the ISO 8601 standard `yyyy-MM-ddTHH:mm:ss.xxxZ`, where:
 - `yyyy-MM-dd` is the four-digit year, followed by the two-digit month and two-digit day.
 - `T` separates the date and the time.
 - `HH:mm:ss.xxx` is the two-digit hour in 24-hour format, followed by the two-digit minute, two-digit second, and three-digit millisecond.
 - `Z` indicates that the time is in Coordinated Universal Time (UTC).

An example is `2025-01-14T16:15:20.123Z`.

- **Number:** Any number.
- **GUID:** A globally unique identifier.
- **Integer:** A whole number that can be positive, negative, or zero.

After a variable has a defined value, if you try to change its type, a consent dialog appears. The dialog alerts you that all the variable values will be reset and that this change could be a breaking change on the consumer item side.

Alternative value sets

When you create a new value set in a variable library, the new values are set as pointers to the default values. You can change them to be a fixed value.

If you change the value of a variable in the alternative value set, the alternative value is saved in the JSON file for value sets. You can change the order in which the value sets appear, or change the value of a variable in the alternative value set, in this JSON file in Git.

Considerations and limitations

Size limitations

- There can be *up to 1,000 variables* and *up to 1,000 value sets*, as long as you meet both of these requirements:
 - The total number of cells in the alternative value sets is less than 10,000.
 - The item's size doesn't exceed 1 MB.

These requirements are validated when you save changes.

- The note field can have up to 2,048 characters.
- The value-set description field can have up to 2,048 characters.

Limitations for alternative value sets

- Alternative value sets in a variable library appear in the order in which you added them. Currently, you can't reorder them in the UI. To change the order, edit the JSON file directly.
- The name of each value set must be unique within a variable library.
- Variable names must be unique within a variable library. You can have two variables with the same name in a workspace if they're in different items.
- There's always one (and only one) active value set in a variable library at a time. You can't delete a value set while it's active. To delete it, first configure another value set to be active. You can have a different active value set for each stage of a deployment pipeline.

Last updated on 12/15/2025

Variable library permissions

This article explains who can access variable libraries and their values.

Permissions for a variable library item

The Microsoft Fabric variable library permissions are aligned with the Fabric workspace model. Permissions are according to your workspace role, or the variable library can be shared directly.

 Expand table

Workspace role	Permissions
Viewer	Can view the variable library item
Contributor	Can view, add, edit, and delete the variable library item
Member	Can view, add, edit, delete, and reshare the variable library item
Admin	Can view, add, edit, delete, and reshare the variable library item

To share a variable library item, go to the item menu in the workspace, and then select **Share**. If the user that you share the item with doesn't have permission to the workspace, but has permission to one of the variable's consumer items (for example, a pipeline), the variable library isn't visible or available for use in that pipeline.

To set an item as a variable value in a variable library, you need to have at least read permission for that item. For example, if you want to set the value of a variable to be a lakehouse, you need read permission for the lakehouse.

For more information about workspace roles, see [Roles in workspaces in Microsoft Fabric](#).

Variable permissions

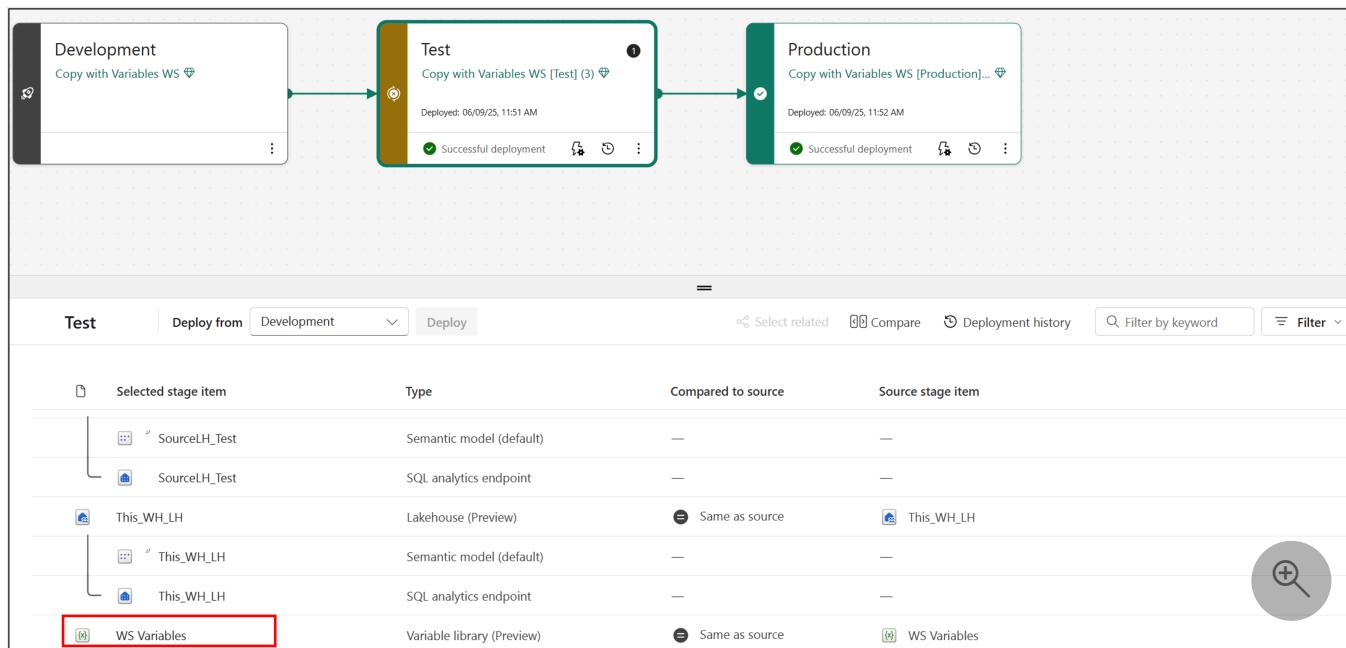
There's no permission management at the variable level. Permission for each variable is the same as the permissions for the entire item.

Variable library CI/CD

You can use Microsoft Fabric variable libraries to manage configurations across stages of the release pipeline and to save values in Git. This article explains how to use variable libraries in the context of lifecycle management and continuous integration and continuous delivery (CI/CD).

Variable libraries and deployment pipelines

You can deploy variable libraries and their values in deployment pipelines to manage variable values across stages.



Remember this important information:

- All *value sets* in the variable library are available to all stages of the deployment pipeline, but only one set is active in a stage.
- The *active value set* for each stage is selected independently. You can change it anytime.
- When you first deploy or commit a variable library, the library's active set has the default value. You can change this value by accessing the newly created variable library in the target stage or repository and changing the active set.

The screenshot shows the 'WS Variables' page. On the left, there's a table of variables with columns for Name, Note, and Type. On the right, there's a section for 'Alternative value sets' with two tabs: 'Test VS' and 'Prod VS'. The 'Prod VS' tab is active, showing a list of value sets. A red box highlights the 'Prod VS' tab and the 'Set as active' button. A magnifying glass icon is in the bottom right corner.

- Although deployments don't affect the *selected active value set* in each stage, you can update the values themselves in the variable library. The consumer item in its workspace (for example, a pipeline) automatically receives the correct value from the active value set.

The following operations to variables or value sets in one stage of a deployment pipeline cause the variable library to be reflected as **Different form source** compared to the same item in a different stage:

- Added, deleted, or edited variables
- Added or deleted value sets
- Names of variables
- Order of variables

Test	Deploy from	Development	Deploy	Select related	Compare	Deployment history
Selected stage item	Type	Compared to source		Source stage item		
SalesVL	Variable library (Preview)		Different from source	SalesVL		

A simple change to the active value set doesn't register as **Different form source** when you compare. The active value set is part of the item configuration, but it's not included in the definition. That's why it doesn't appear on the deployment pipeline comparison and isn't overwritten on each deployment.

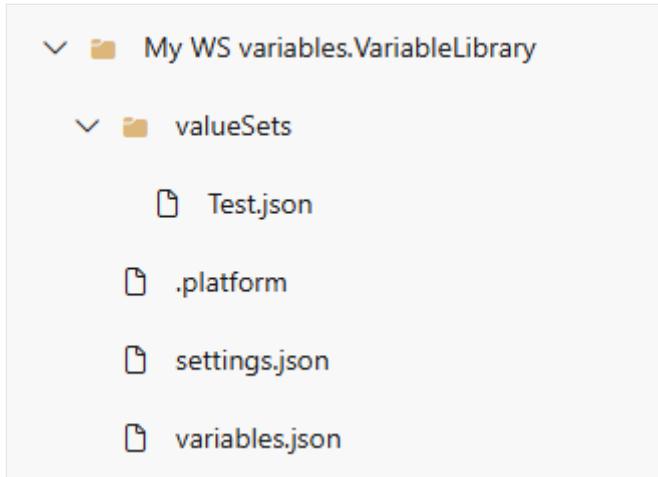
Variable libraries and Git integration

Like other Fabric items, variable libraries can be integrated with Git for source control. Variable library items are stored as folders that you can maintain and sync between Fabric and your Git provider.

Item permissions are checked during Git update and commit.

The schema for the variable library item is a JSON object that contains four parts:

- Folder for value sets
- Settings
- [Platform.json](#), an automatically generated file
- Variables



Value sets

The variable library folder contains a subfolder called `valueSets`. This folder contains a JSON file for each value set. This JSON file contains only the variable values for *non-default* values in that value set.

For more information about the value set file, see the [value set example](#).

Values for variables not in this file are taken from the default value set.

Settings

The `settings.json` file contains settings for the variable library.

For more information, see the [settings.json example](#).

Variables

The `variables.json` file contains the variable names and their default values.

For more information, see the [variables.json example](#).

Considerations and limitations

Size limitations

- There can be *up to 1,000 variables* and *up to 1,000 value sets*, as long as you meet both of these requirements:
 - The total number of cells in the alternative value sets is less than 10,000.
 - The item's size doesn't exceed 1 MB.

These requirements are validated when you save changes.

- The note field can have up to 2,048 characters.
- The value-set description field can have up to 2,048 characters.

Limitations for alternative value sets

- Alternative value sets in a variable library appear in the order in which you added them. Currently, you can't reorder them in the UI. To change the order, edit the JSON file directly.
- The name of each value set must be unique within a variable library.
- Variable names must be unique within a variable library. You can have two variables with the same name in a workspace if they're in different items.
- There's always one (and only one) active value set in a variable library at a time. You can't delete a value set while it's active. To delete it, first configure another value set to be active. You can have a different active value set for each stage of a deployment pipeline.

Related content

- [Git integration source code format](#)

Last updated on 12/15/2025

Automate variable libraries by using APIs

You can use the [Microsoft Fabric REST APIs](#) to fully automate the management of variable libraries in application lifecycle management (ALM).

If you're using the APIs as part of your lifecycle management, permissions for item reference are checked during Git update and pipeline deployment.

Prerequisites

To use the APIs, you need:

- The same prerequisites as for the [variable library item](#).
- A Microsoft Entra token for the Fabric service. Use the token in the authorization header of the API call. For information about how to get a token, see [Fabric API quickstart](#).

Variable library APIs

You can use the [variable library REST APIs](#) to perform the following functions:

- [Create a variable library](#): Create a variable library in the specified workspace.
- [Get a variable library](#): Retrieve properties of a variable library. You can also retrieve the active value set by using the `VariableLibraryProperties` parameter.
- [Update a variable library](#): Update a variable library's properties. You can also update the active value set by using the `VariableLibraryProperties` parameter.
- [Delete a variable library](#): Delete the specified variable library.
- [List variable libraries](#): List variable libraries in the specified workspace.
- [Get a variable library's definition](#): Retrieve the public [definition](#) of a variable library.
- [Update a variable library's definition](#): Override the [definition](#) of a variable library.

The variable library REST APIs support service principals.

Examples

For some examples of how to use the APIs, see the [REST documentation](#) for each API.

For a breakdown of the definition structure of a variable library, see [Variable library definition](#).

Considerations and limitations

Size limitations

- There can be *up to 1,000 variables* and *up to 1,000 value sets*, as long as you meet both of these requirements:
 - The total number of cells in the alternative value sets is less than 10,000.
 - The item's size doesn't exceed 1 MB.

These requirements are validated when you save changes.

- The note field can have up to 2,048 characters.
- The value-set description field can have up to 2,048 characters.

Limitations for alternative value sets

- Alternative value sets in a variable library appear in the order in which you added them. Currently, you can't reorder them in the UI. To change the order, edit the JSON file directly.
- The name of each value set must be unique within a variable library.
- Variable names must be unique within a variable library. You can have two variables with the same name in a workspace if they're in different items.
- There's always one (and only one) active value set in a variable library at a time. You can't delete a value set while it's active. To delete it, first configure another value set to be active. You can have a different active value set for each stage of a deployment pipeline.

Last updated on 12/15/2025

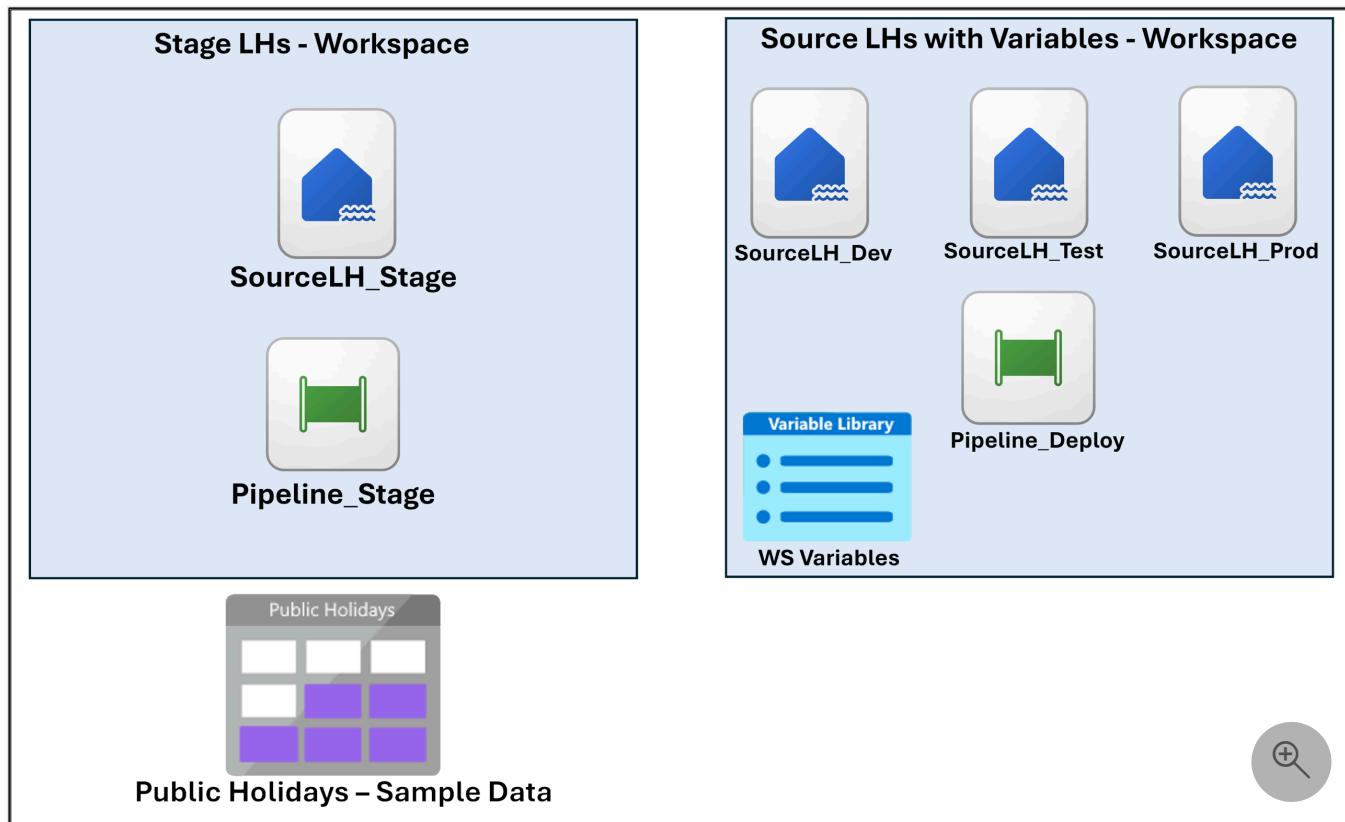
Tutorial: Use variable libraries to customize and share item configurations

This tutorial shows you how to use dynamic content in Microsoft Fabric pipelines. When you create a variable library item and add variables to it, you can automate values for various stages of your deployment pipeline. In this tutorial, you copy data from one lakehouse to another. Then you use the variable library to set the source and destination values for the copy activity.

In this tutorial, you:

- ✓ Create a variable library.
- ✓ Add variables to the library.
- ✓ Define additional value sets for the variables.
- ✓ Consume the variables in another item in the workspace (a pipeline).
- ✓ Edit the variables in a Git repository.
- ✓ Create a deployment pipeline and deploy the variable library.
- ✓ Change the active value set in the target stage of the deployment pipeline.
- ✓ Show that the value of the variable complies with the active value set in each stage.

The following diagram shows the workspace layout for this tutorial.



Prerequisites

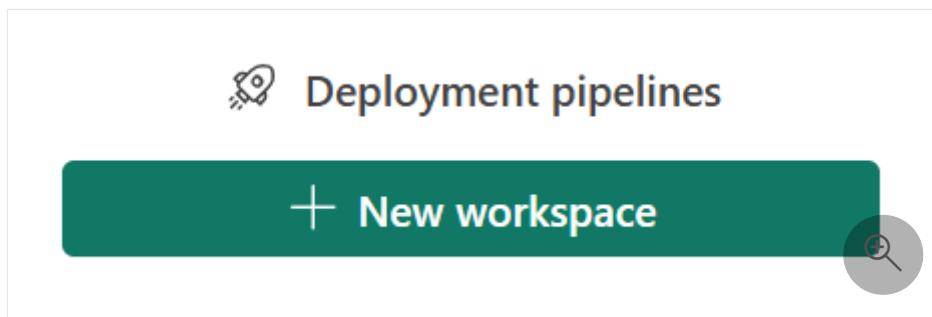
- A Fabric tenant account with an active subscription. [Create an account for free.](#)
- The following [tenant switch](#) is enabled from the admin portal:
 - [Users can create Fabric items](#)

The tenant admin, capacity admin, or workspace admin can enable these switches, depending on your [organization's settings](#).

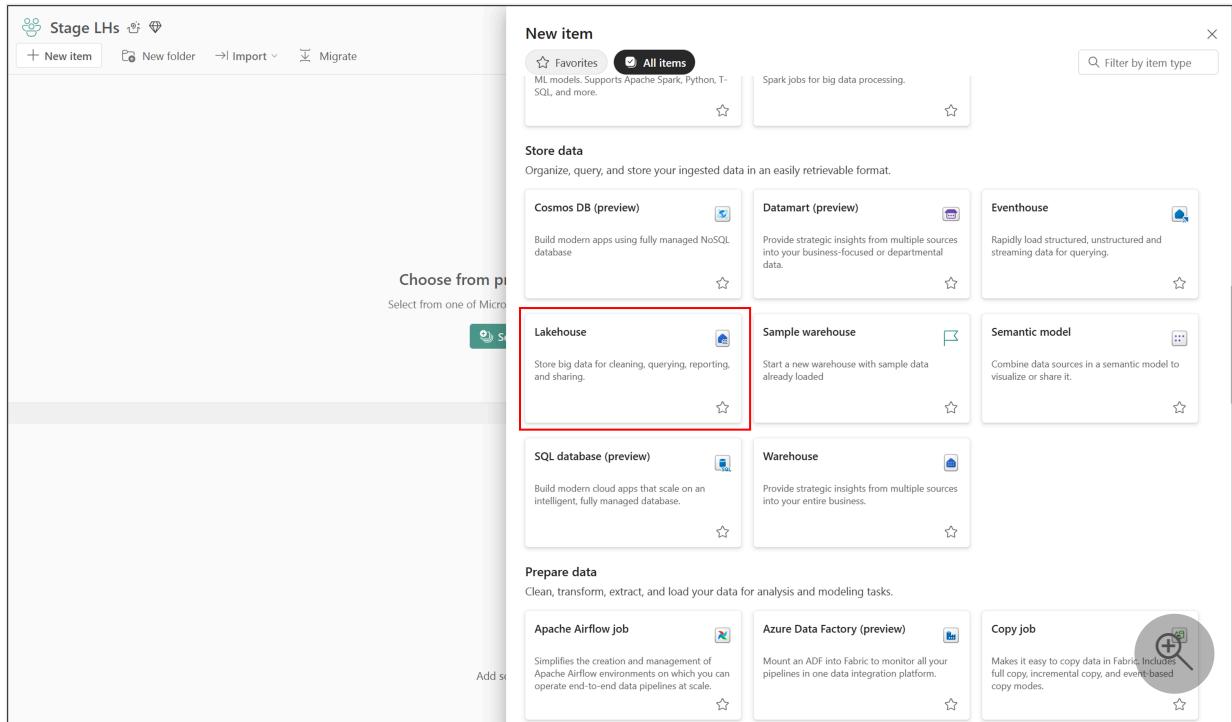
Create the Stage LHS workspace, SourceLH_Stage lakehouse with sample data, and Pipeline_Stage pipeline

First, create a workspace and lakehouse to use as your initial staging data:

1. Go to [Power BI](#).
2. On the sidebar, select **Workspace**.
3. [Create a workspace](#). Call it **Stage LHS**.



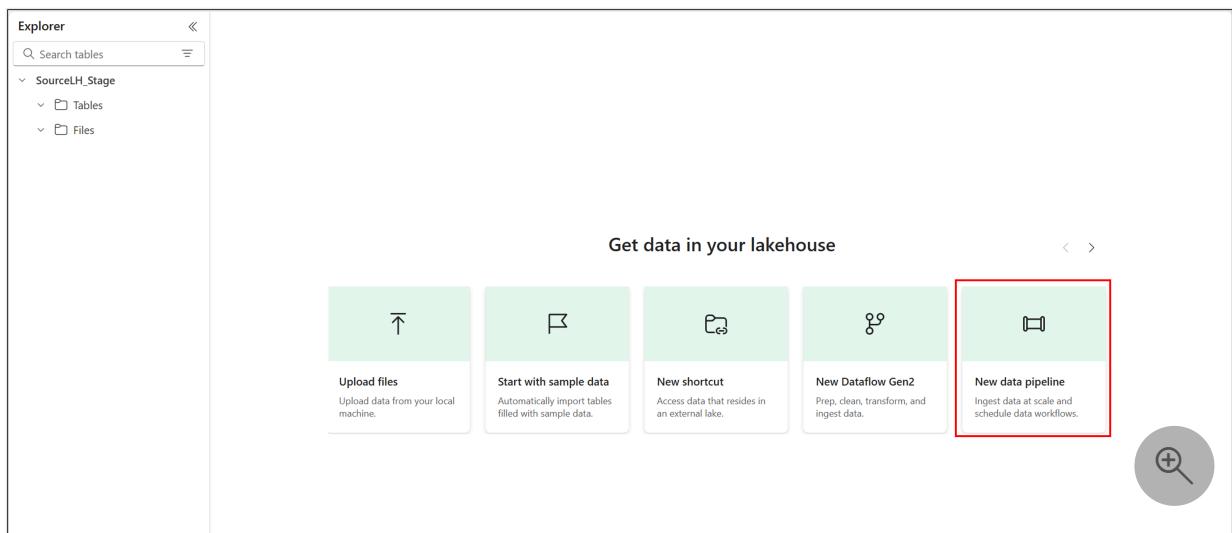
4. Create a lakehouse:
 - a. At the top of the workspace, select **New item**.
 - b. Under **Store data**, select **Lakehouse**.



c. Enter the name **SourceLH_Stage**, and then select **Create**.

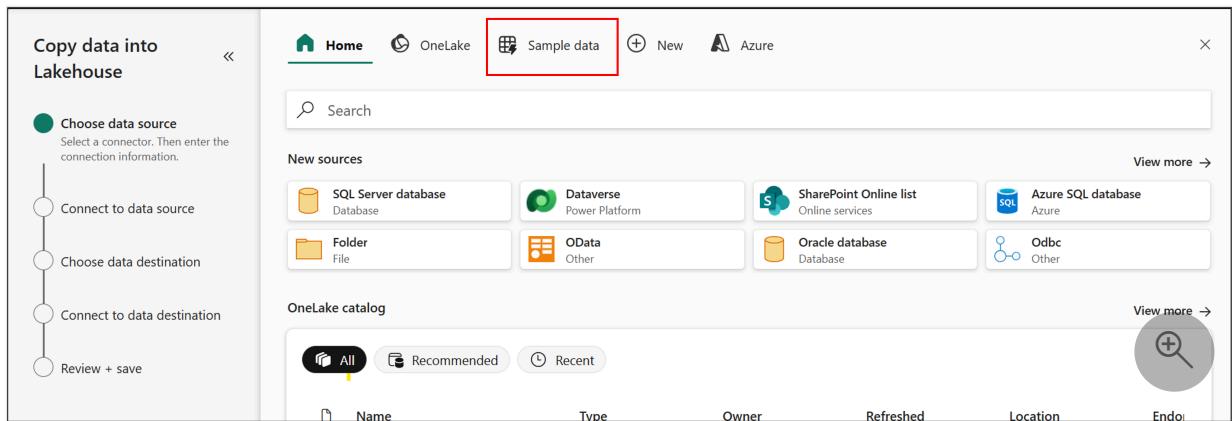
5. Create a pipeline:

a. In the lakehouse, select **New pipeline**.

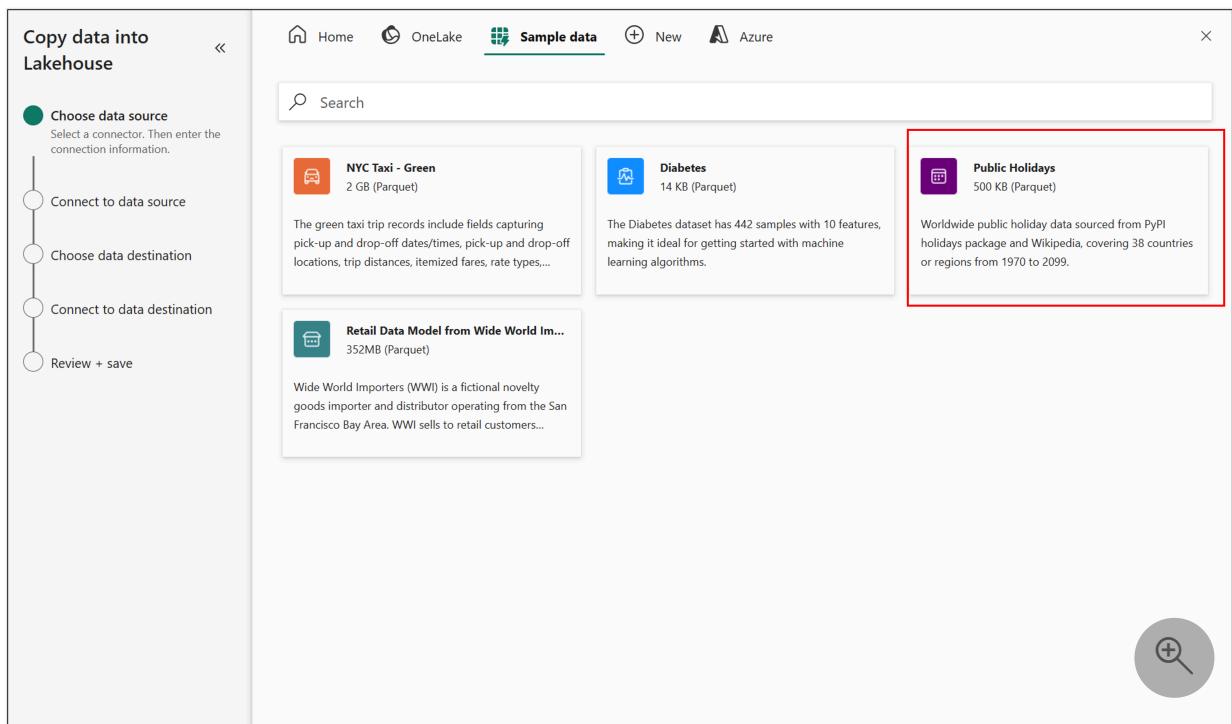


b. Enter the name **Pipeline_Stage**, and then select **Create**.

c. In the **Copy data into Lakehouse** wizard, on the **Choose data source** page, select **Sample data**.



d. Select Public Holidays.



e. After the sample data finishes loading, select Next.

f. On the Connect to data destination page, select Next.

Copy data into Lakehouse

Connect to data destination
Select and map to folder path or table.

Connection: SourceLH_Staged
Root folder: Tables
Load settings: Load to new table
Table: Processed

Column mappings:

Source	Type	Destination	Type
countryOrRegion	abc STRING	countryOrRegion	abc string
holidayName	abc STRING	holidayName	abc string
normalizeHolidayName	abc STRING	normalizeHolidayName	abc string
isPaidTimeOff	✗ BOOLEAN	isPaidTimeOff	✗ boolean

Enable partitions

Back **Next**

g. On the Review + save page, select Save + Run.

Copy data into Lakehouse

Review + save
Confirm Copy summary

Copy Summary

Source: Public Holidays

Destination: Microsoft Fabric Lakehouse T...

Options: Start data transfer immediately

Back **Save + Run**

Create the Source LHS with Variables workspace

Now, create the workspace that you'll work out of and use with your variable library:

1. Go to [Power BI](#).
2. On the sidebar, select **Workspace**.
3. [Create a workspace](#). Call it **Source LHS with Variables**.

Create the SourceLH_Dev, SourceLH_Test, and SourceLH_Prod lakehouses

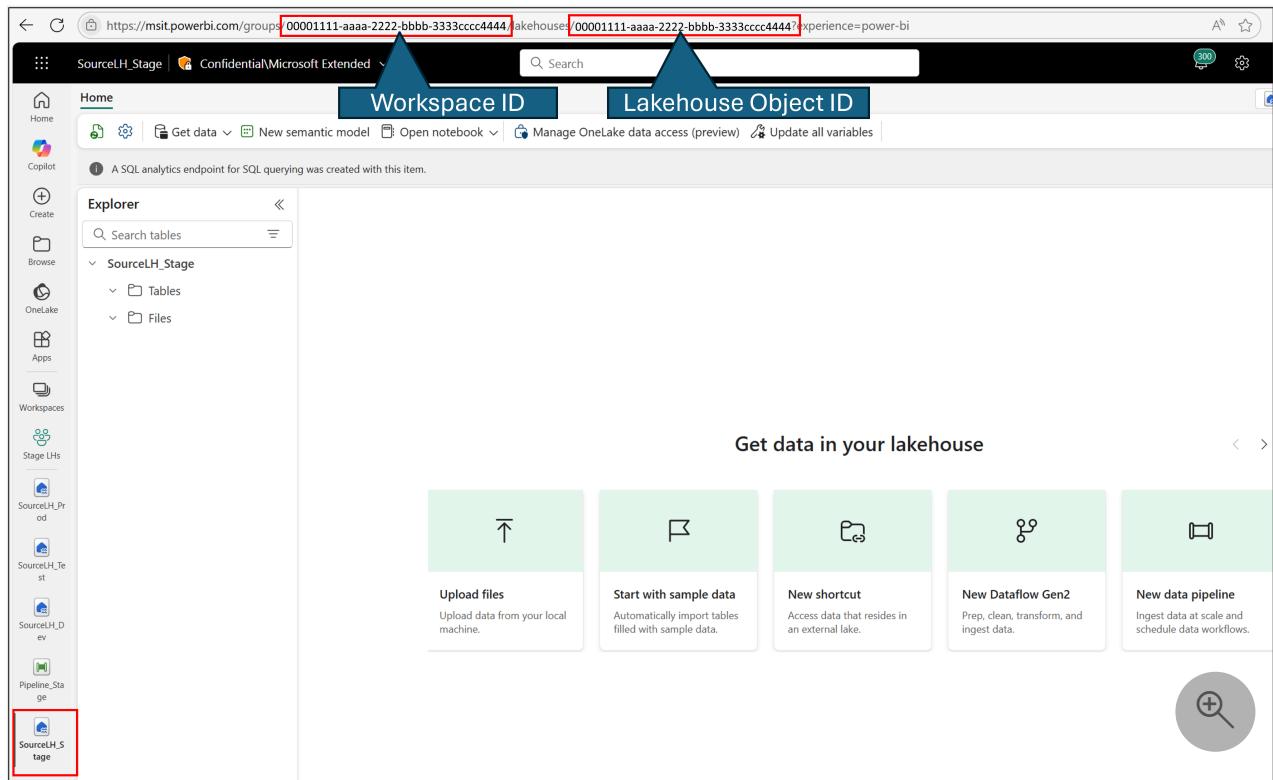
Next, create the three lakehouses to use with the variable library:

1. Create the first lakehouse:
 - a. On the sidebar, select the **Source LHs with Variables** workspace.
 - b. Select **New item**.
 - c. Under **Store data**, select **Lakehouse**.
 - d. Enter the name **SourceLH_Dev**, and then select **Create**.
2. Create the second lakehouse by following the preceding steps. Name it **SourceLH_Test**.
3. Create the third lakehouse by following the preceding steps. Name it **SourceLH_Prod**.
4. On the sidebar, select the **Source LHs with Variables** workspace and make sure that it contains all the newly created lakehouses.

Get the workspace IDs and object IDs for lakehouses

In these steps, you get the unique identifiers to use in your variable library:

1. In [Power BI](#), on the sidebar, select the **Stage LHs** workspace.
2. In the workspace, select the **SourceLH_Stage** lakehouse.
3. Copy the workspace ID and the lakehouse object ID in the URL.



4. Repeat the preceding steps for the **SourceLH_Dev** and **SourceLH_Test** lakehouses in the **Source LHS with Variables** workspace.

Create a variable library with variables

Now, create the variable library:

1. In the **Source LHS with Variables** workspace, select **New item**.
2. Under **Develop data**, select ****Variable library ****.

The screenshot shows the Microsoft Fabric 'Choose from platform' interface. On the left, there's a sidebar with a tree view of workspace items categorized by type (Lakehouse, Semantic model, etc.). On the right, there are several cards representing different features:

- New item**: streaming data for querying.
- Scorecard**: Define, track, and share key goals for your organization.
- Develop data**: Create and build your software, applications, and data solutions.
 - API for GraphQL**: Create an API for GraphQL to easily connect your applications to Fabric data sources.
 - Environment**: Set up shared libraries, Spark compute settings, and resources for notebooks and Spark job definitions.
 - Notebook**: Explore, analyze, and visualize data and build ML models. Supports Apache Spark, Python, T-SQL, and more.
 - Variable library (preview)**: Create item variables that can be referenced across this workspace. This card is highlighted with a red box.
 - User data functions (preview)**: Author, host, and manage serverless user data functions optimized for Fabric.
- Others**: Find unique or third-party provided functionality that builds on Fabric's core capabilities.
 - Event Schema Set (preview)**: Define the structure and format of events through schemas, enabling data quality and validation with group-level access control.
 - Healthcare data solutions**: Use advanced AI analytics to help close care gaps, generate new insights, enhance patient care, and improve outcomes.
 - Metric set (preview)**: Centralize your metrics so others can explore and re-use trusted data at their own pace.

3. Name the library **WS variables**, and then select **Create**.

4. Select **New variable**.

The screenshot shows the 'WS variables' page. At the top, there are buttons for Save, + New variable, Delete variable, and Add value set. Below this is a search bar and filter options. The main area displays a large placeholder icon with a paperclip and the text 'There are no variables'. A link to 'Learn more' is present. At the bottom is a 'New variable' button and a magnifying glass icon.

5. Create the following variables:

Expand table

Name	Type	Default value set
Source_LH	String	<GUID of SourceLH_Stage lakehouse>
Source_WSID	String	<GUID of SourceLH_Stage workspace>

Name	Type	Default value set
Destination_LH	String	<GUID of SourceLH_Dev lakehouse>
Destination_WSID	String	<GUID of SourceLH_Dev workspace>
SourceTable_Name	String	Processed
DestinationTable_Name	String	DevCopiedData

WS Variables

Variables

Name *	Type	Note	Default value set *	Active	...
Source_LH	String	<input type="button" value="Edit"/>	1f61c499-89cd-4df5-92c7-d110b...		
Source_WSID	String	<input type="button" value="Edit"/>	71bc08cb-a3dd-4d72-b101-1b3af...		
Destination_LH	String	<input type="button" value="Edit"/>	4fe228d3-a363-4b7f-a5d4-fae9d2...		
Destination_WSID	String	<input type="button" value="Edit"/>	dfdf8621-3a7f-44ed-a44d-64ae48...		
SourceTable_Name	String	<input type="button" value="Edit"/>	Processed		
DestinationTable_Name	String	<input type="button" value="Edit"/>	DevCopiedData		<input type="button" value="Search"/>

6. Select Save.

Create alternate value sets

In these steps, you add the alternate value sets to your variable library:

1. Create the first value set:
 - a. In the WS Variables variable library, select Add value set.
 - b. Enter Test VS for the name, and then select Create.
 - c. Create the following variables:

 Expand table

Name	Type	Default value set
Source_LH	String	<GUID of SourceLH_Dev lakehouse>
Source_WSID	String	<GUID of SourceLH_Dev workspace>
Destination_LH	String	<GUID of SourceLH_Test lakehouse>
Destination_WSID	String	<GUID of SourceLH_Test workspace>
SourceTable_Name	String	DevCopiedData
DestinationTable_Name	String	TestCopiedData

d. Select **Save > Agree.**

2. Create the second value set:

a. Select **Add value set.**

b. Enter **Prod VS** for the name, and then select **Create.**

c. Create the following variables:

 Expand table

Name	Type	Default value set
Source_LH	String	<GUID of SourceLH_Test lakehouse>
Source_WSID	String	<GUID of SourceLH_Test workspace>
Destination_LH	String	<GUID of SourceLH_Prod lakehouse>
Destination_WSID	String	<GUID of SourceLH_Prod workspace>
SourceTable_Name	String	TestCopiedData
DestinationTable_Name	String	ProdCopiedData

The screenshot shows the 'WS Variables' workspace. On the left, there is a table of variables with columns for Name, Note, Type, and Value. On the right, there are two sections: 'Alternative value sets' for 'Test VS' and 'Prod VS', each listing several items with their corresponding values.

Name	Type	Value
Source_LH	String	1f61c499-89cd-4df5-92c7-d110b...
Source_WSID	String	71bc08cb-a3dd-4d72-b101-1b3af...
Destination_LH	String	4fe228d3-a363-4b7f-a5d4-fae9d2...
Destination_WSID	String	dffdf8621-3a7f-44ed-a44d-64ae48...
SourceTable_Name	String	Processed
DestinationTable_Name	String	DevCopiedData

Alternative value sets

- Test VS**
- Prod VS**

d. Select Save > Agree.

Create the Pipeline_Deploy pipeline and declare variables

In these steps, you create your pipeline and declare your variables:

1. In the **Source LHS with Variables** workspace, select **New item**.
2. Under **Get data**, select **Pipeline**.

The screenshot shows the 'Source LHS with Variables' workspace. On the left, there is a list of existing pipelines. On the right, the 'New item' dialog is open, showing various item types like Report, Scorecard, Copy job, Data pipeline, etc. The 'Data pipeline' option is highlighted with a red box.

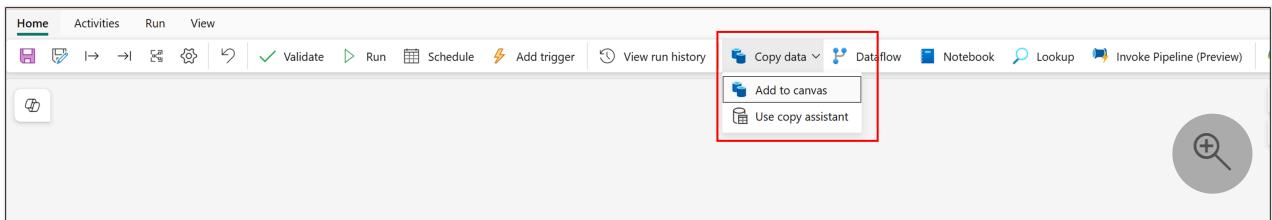
New item

Get data

Data pipeline

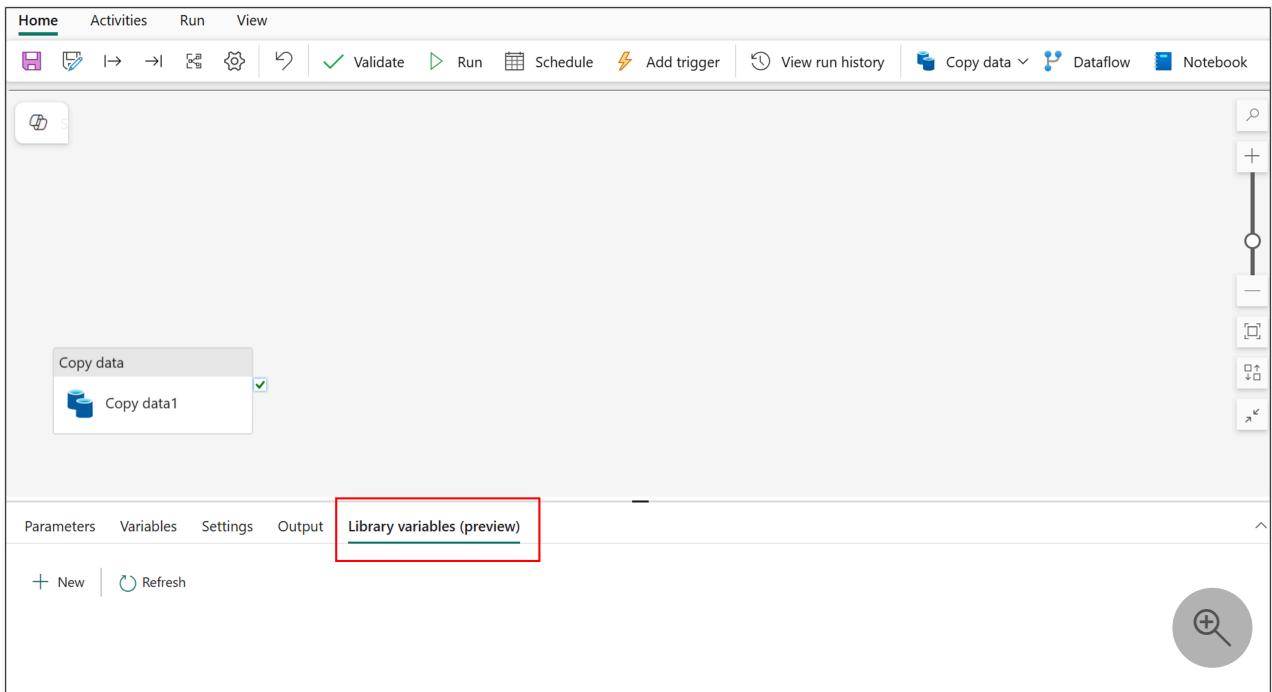
3. Enter the name **Pipeline_Deploy**, and then select **Create**.

4. Select **Copy data > Add to canvas**.



5. Select the canvas so that the focus is off **Copy data**.

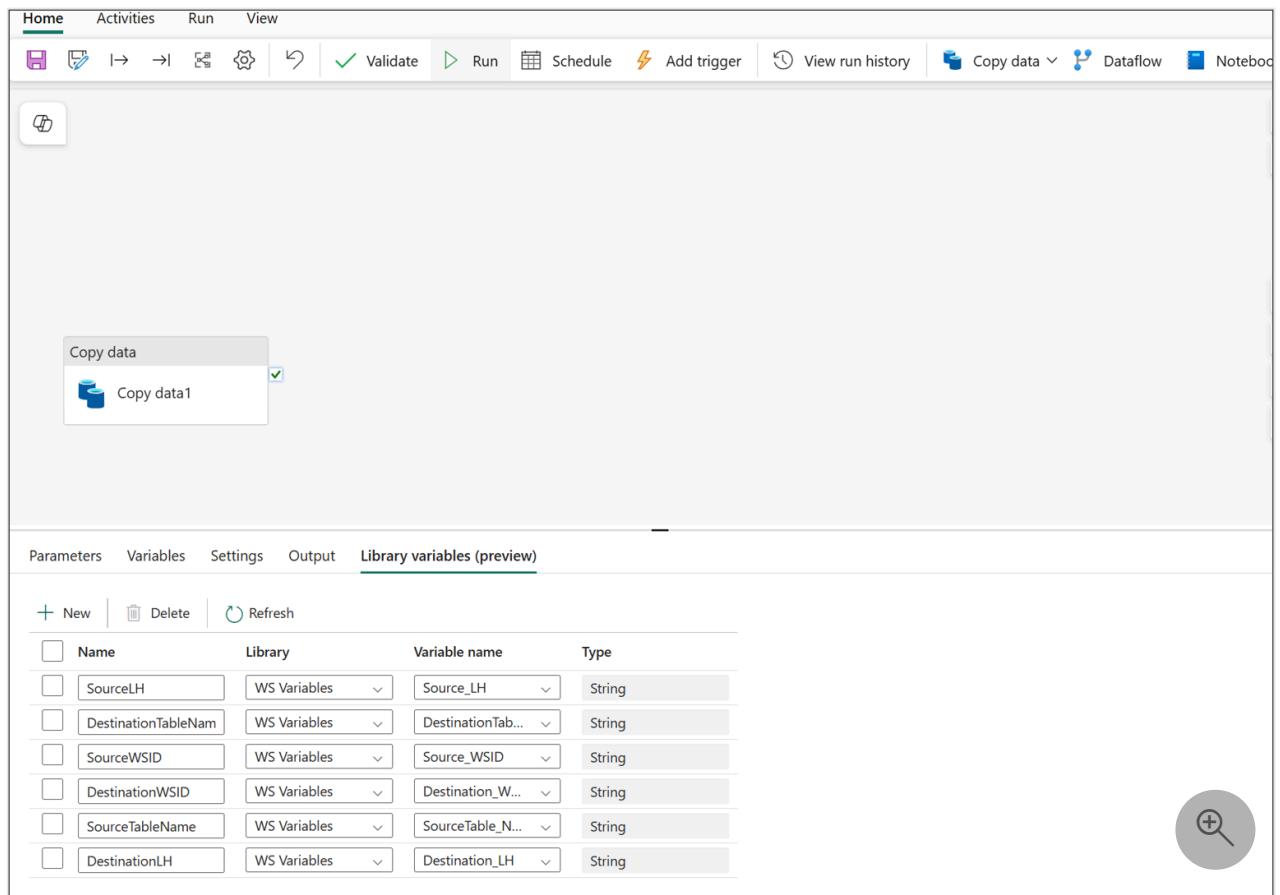
6. Select ****Library variables ****.



7. Select **New**, and then add the following variables:

Expand table

Name	Library	Variable name	Type
SourceLH	WS Variables	Source_LH	String
SourceWSID	WS Variables	Source_WSID	String
DestinationLH	WS Variables	Destination_LH	String
DestinationWSID	WS Variables	Destination_WSID	String
SourceTableName	WS Variables	SourceTable_Name	String
DestinationTableName	WS Variables	DestinationTable_Name	String

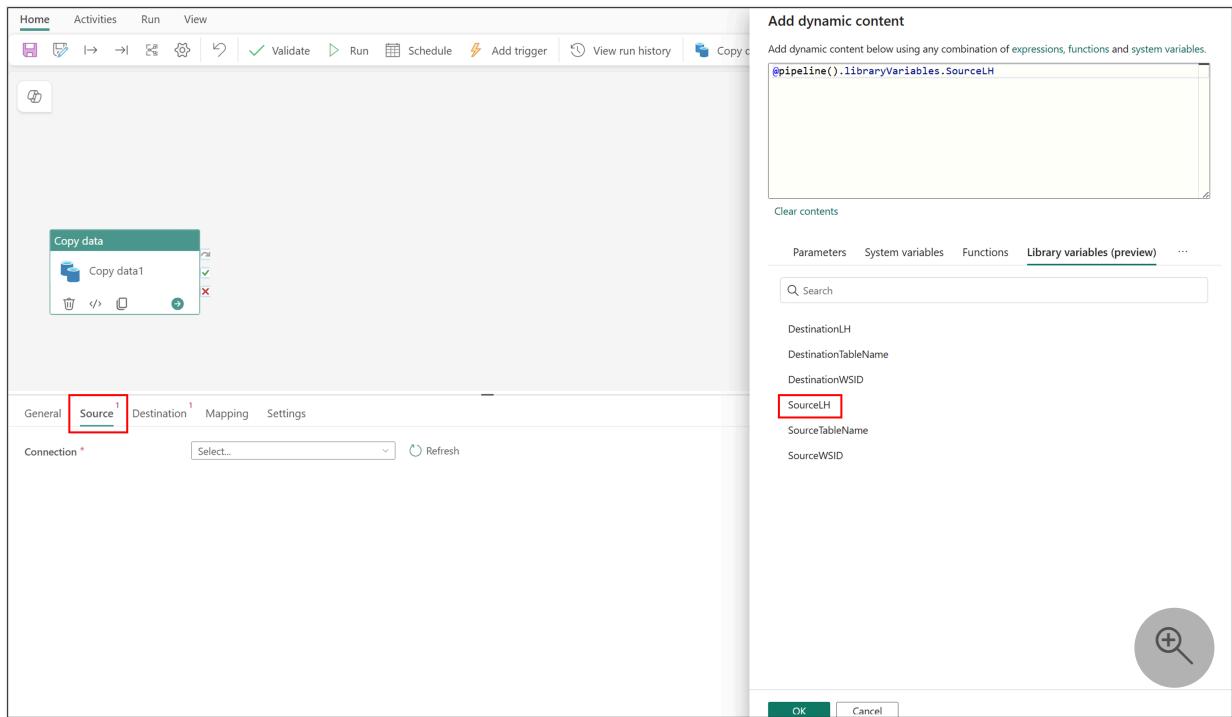


8. Select Save.

Configure the source connection for the Pipeline_Deploy pipeline

In these steps, you configure the source connection for your pipeline:

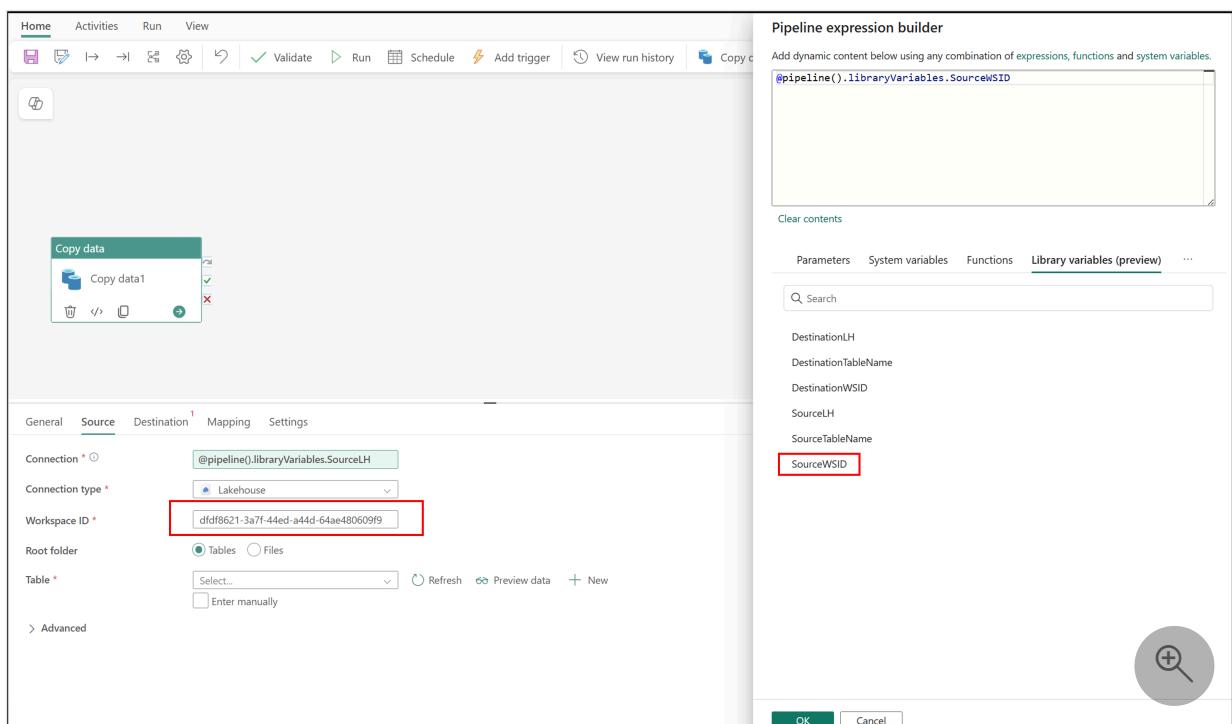
1. In the **Source LHs with Variables** workspace, go to **Pipeline_Deploy**.
2. On the canvas, select **Copy data** so that the focus is on **Copy data**.
3. Select **Source**.
4. Configure **SourceLH**:
 - a. Under **Source > Connection**, select **Add dynamic content**.
 - b. Select the ellipsis (...), and then select ****Library variables ****.
 - c. Select **SourceLH**. It populates the box with `@pipeline().libraryVariables.SourceLH`. Select **OK**.



5. Configure SourceWSID:

- Under **Source > Workspace ID**, select **Add dynamic content**.
- Select the ellipsis (...), and then select ****Library variables ****.
- Select **SourceWSID**. It populates the box with

`@pipeline().libraryVariables.SourceWSID`. Select **OK**.



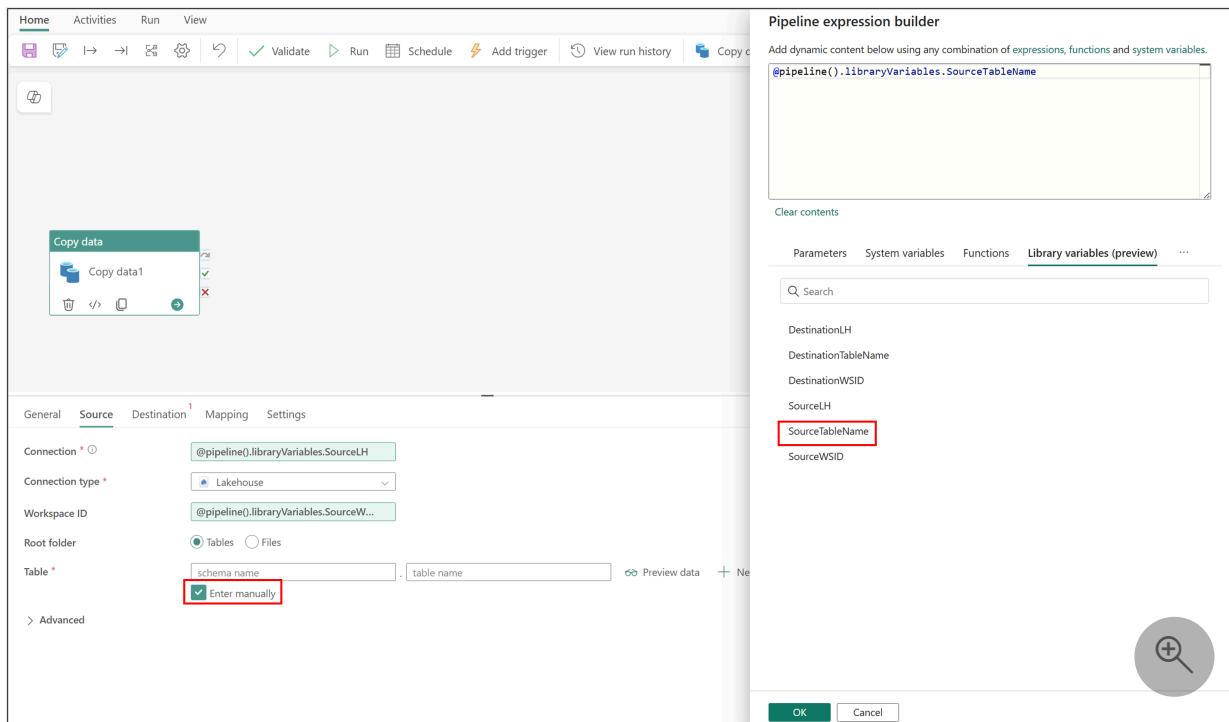
6. Configure SourceTableName:

a. Under Source > Table, select Enter manually, select Table name, and then select Add dynamic content.

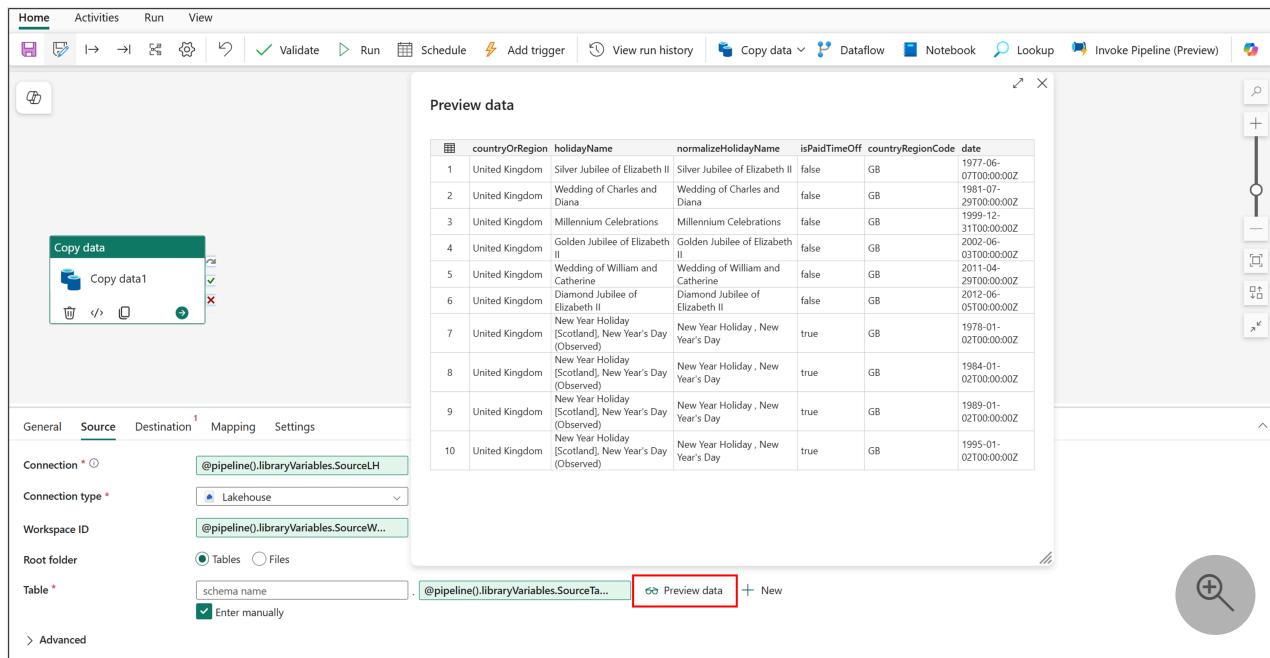
b. Select the ellipsis (...), and then select **Library variables **.

c. Select **SourceTableName**. It populates the box with

`@pipeline().libraryVariables.SourceTableName`. Select OK.



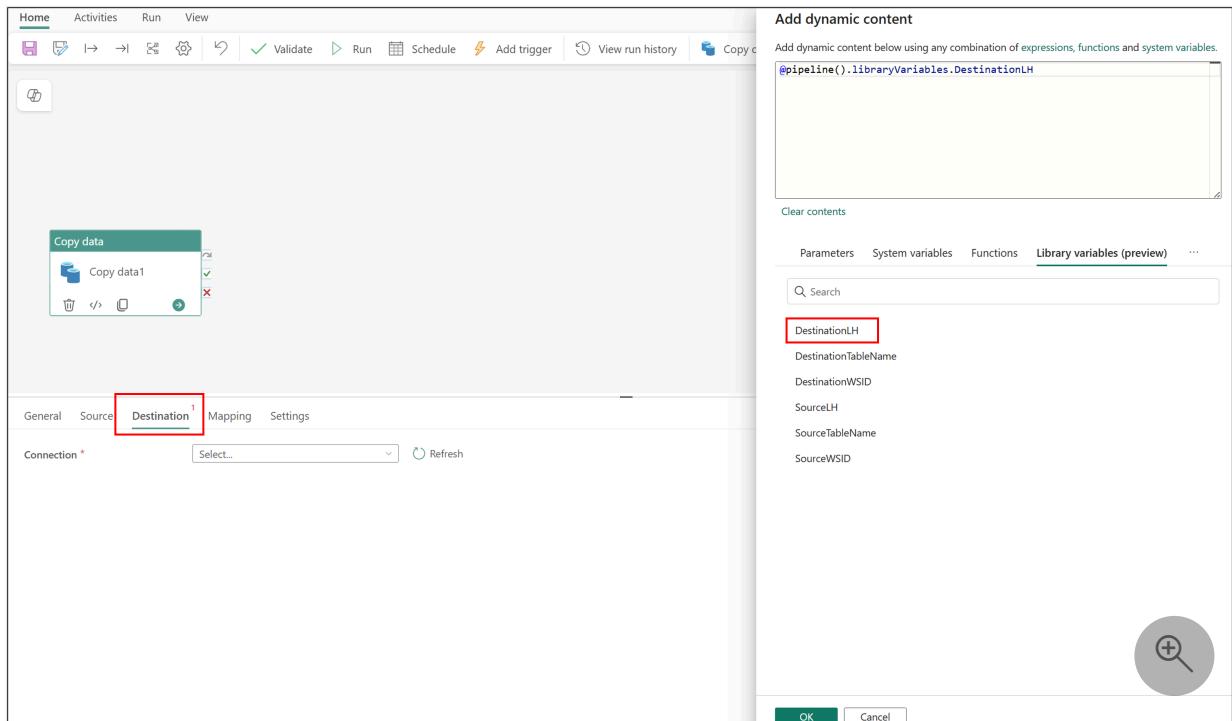
7. Now that the source connection is set up, you can test it. Select **Preview data**, and then select **OK** on the flyout. After the data is populated, you can close the data preview.



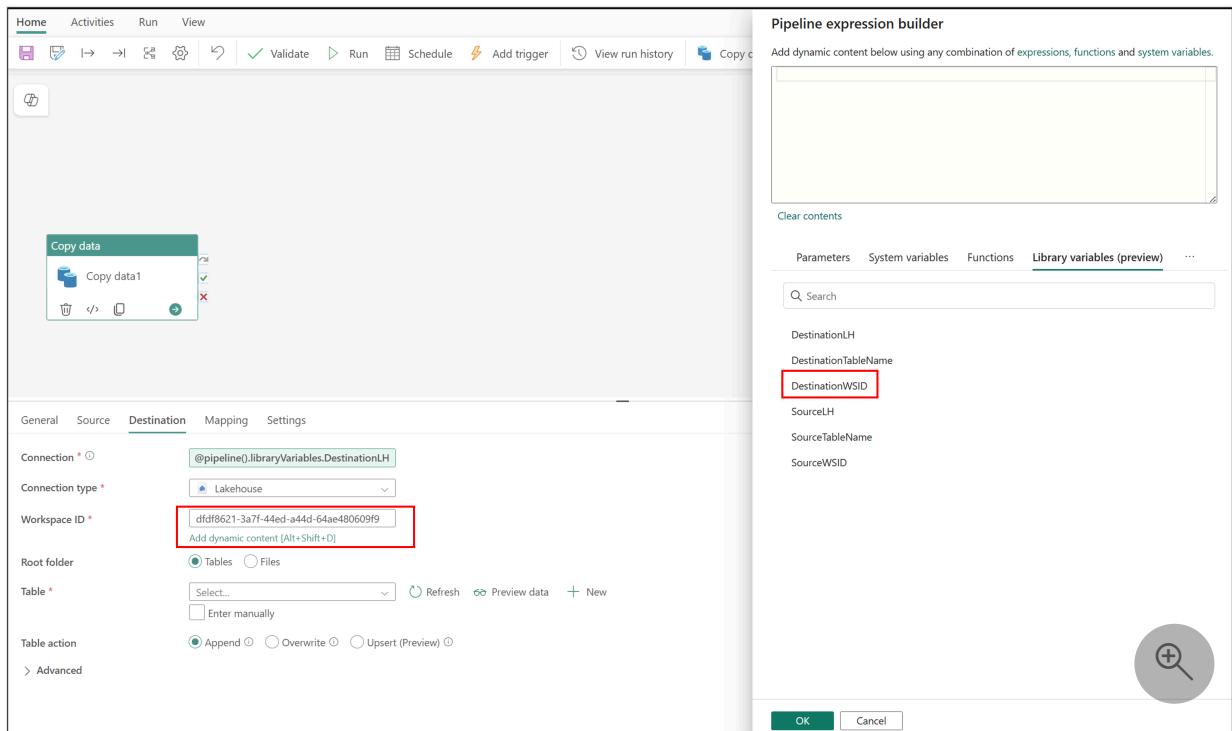
Configure the destination connection for the Pipeline_Deploy pipeline

In these steps, you configure the destination connection for your pipeline:

1. In the **Source LHs with Variables** workspace, go to **Pipeline_Deploy**.
2. On the canvas, select **Copy data** so that the focus is on **Copy data**.
3. Select **Destination**.
4. Configure **SourceLH**:
 - a. Under **Destination > Connection**, select **Add dynamic content**.
 - b. Select the ellipsis (...), and then select ****Library variables ****.
 - c. Select **SourceLH**. It populates the box with
`@pipeline().libraryVariables.DestinationLH`. Select **OK**.

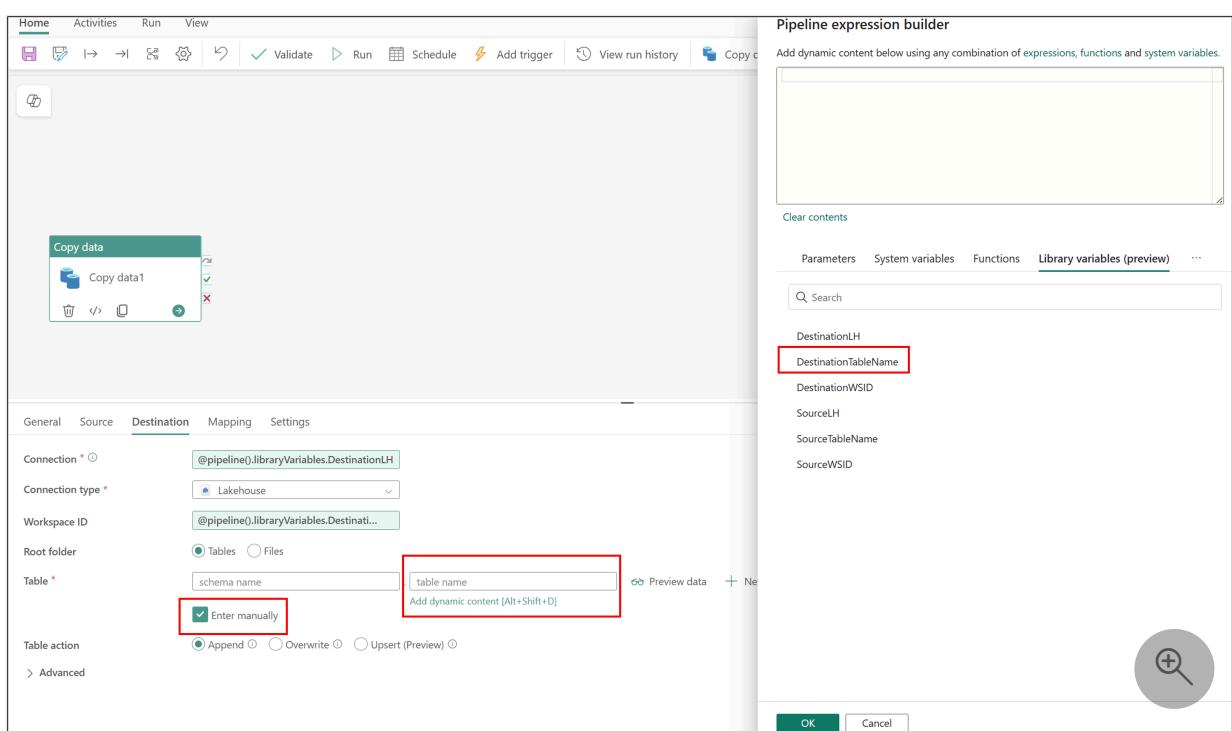


5. Configure **DestinationWSID**:
 - a. Under **Destination > Workspace ID**, select **Add dynamic content**.
 - b. Select the ellipsis (...), and then select ****Library variables ****.
 - c. Select **DestinationWSID**. It populates the box with
`@pipeline().libraryVariables.DestinationWSID`. Select **OK**.

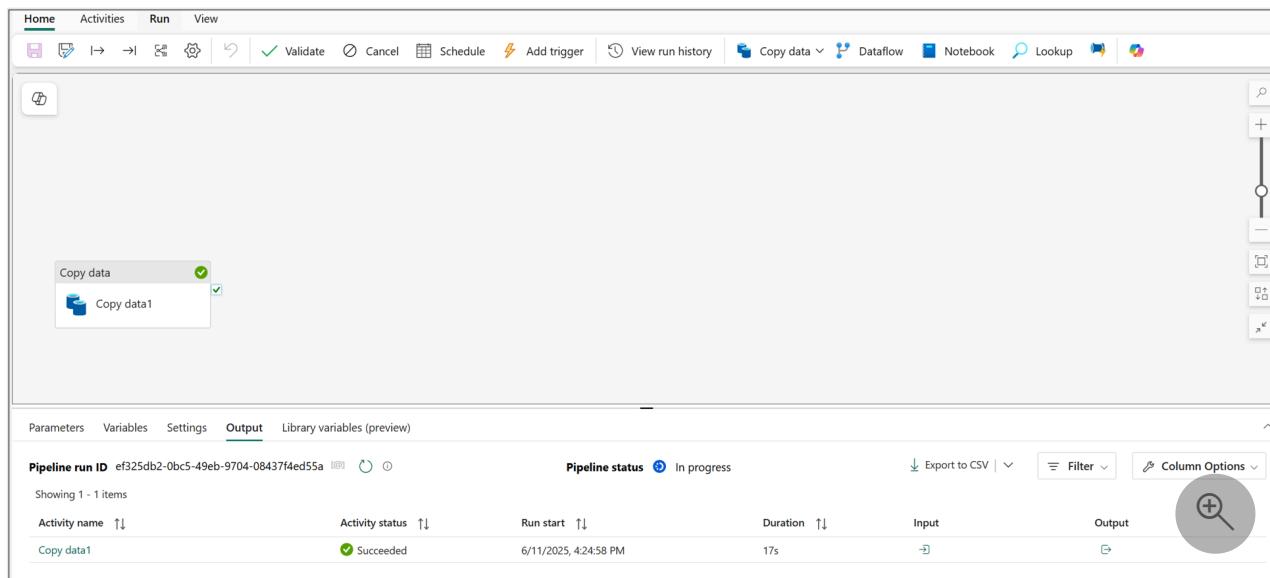


6. Configure DestinationTableName:

- Under **Destination > Table**, select **Enter manually**, select **Table name**, and then select **Add dynamic content**.
- Select the ellipsis (...), and then select ****Library variables ****.
- Select **DestinationTableName**. It populates the box with **@pipeline().libraryVariables.DestinationTableName**. Select **OK**.



7. Now that the destination connection is set up, save the pipeline and select Run. Confirm that it successfully runs.



Create the deployment pipeline

Now, create your deployment pipeline:

1. In the **Source LHS with Variables** workspace, select **Create deployment pipeline**.
2. Name the pipeline **Deployment_Pipeline_Var**, and then select **Next**.

Add a new deployment pipeline

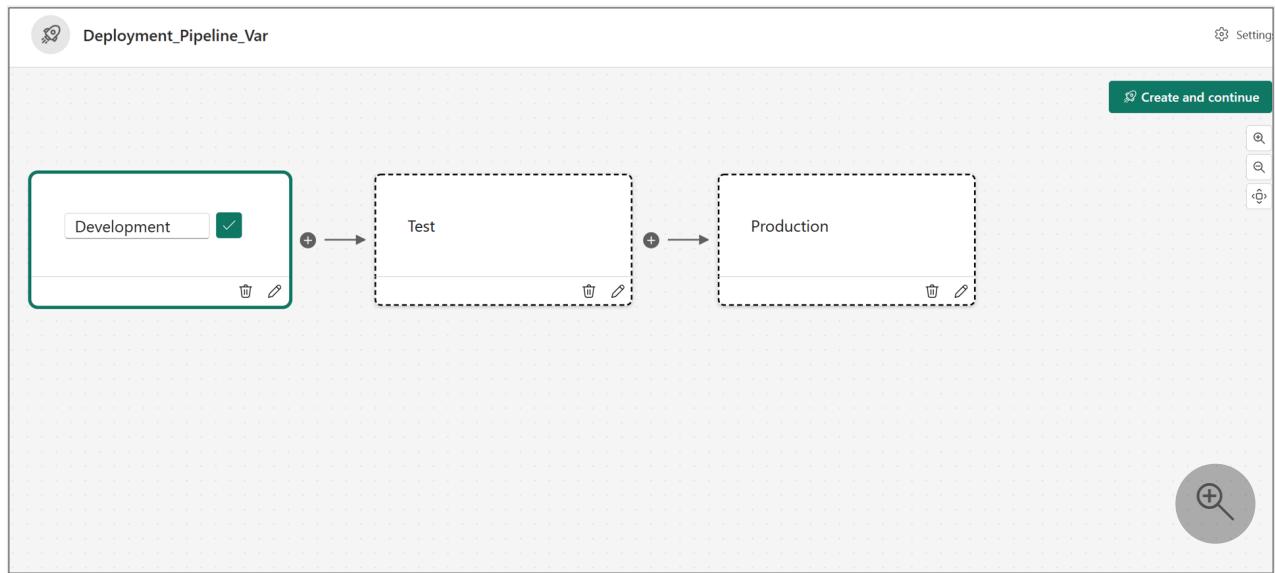
Use this pipeline to manage the content of your workspace or app.

Pipeline name

Description (optional)

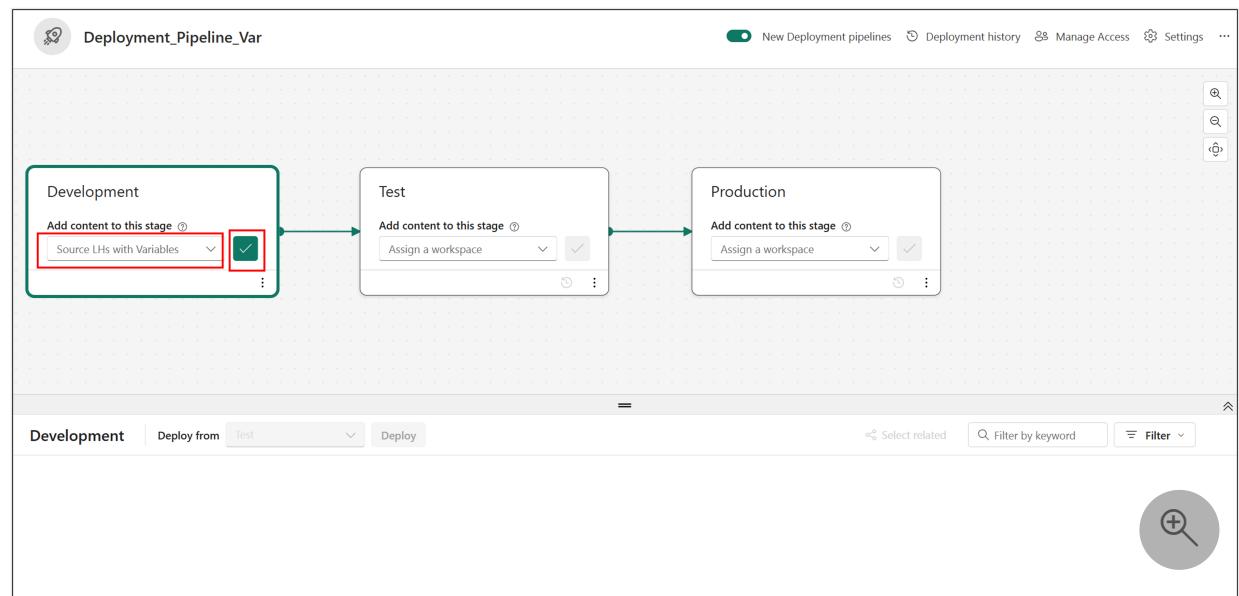
Next **Cancel**

3. In the deployment pipeline, select **Create and continue**.

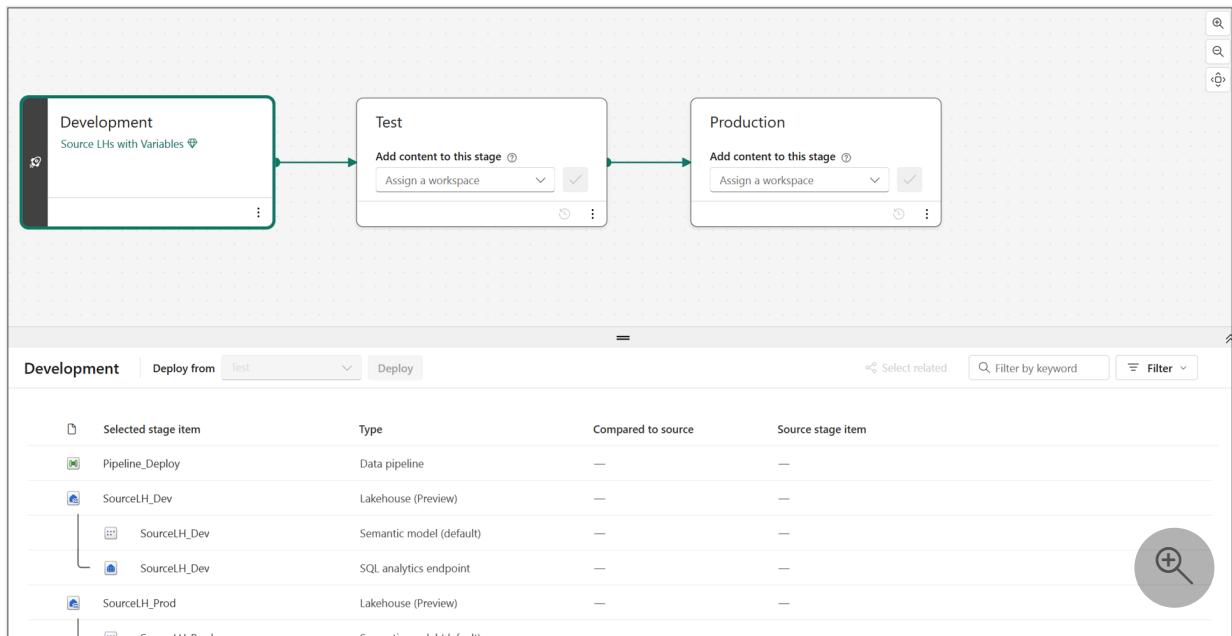


4. For the **Development** stage:

- In the dropdown list, select **Source LHS with Variables** for the workspace. Then select the **Assign** check mark.

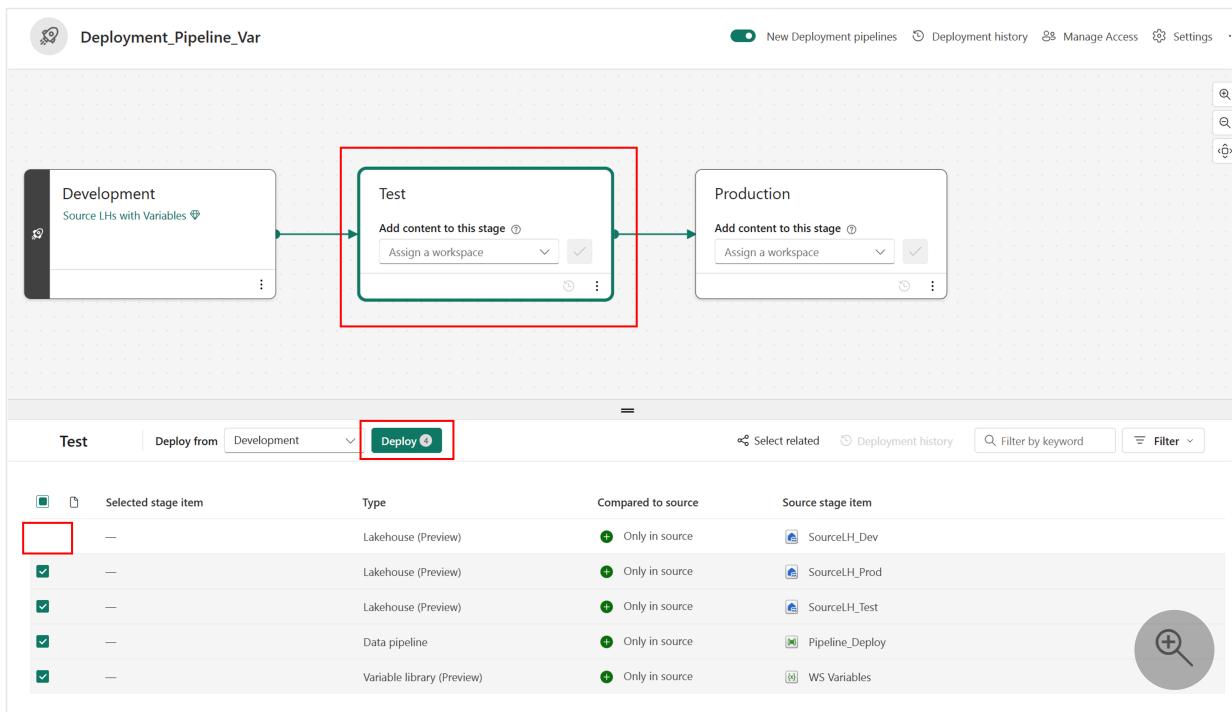


- Select **Continue**. The stage should now be populated with the items from the workspace.



5. For the **Test** stage:

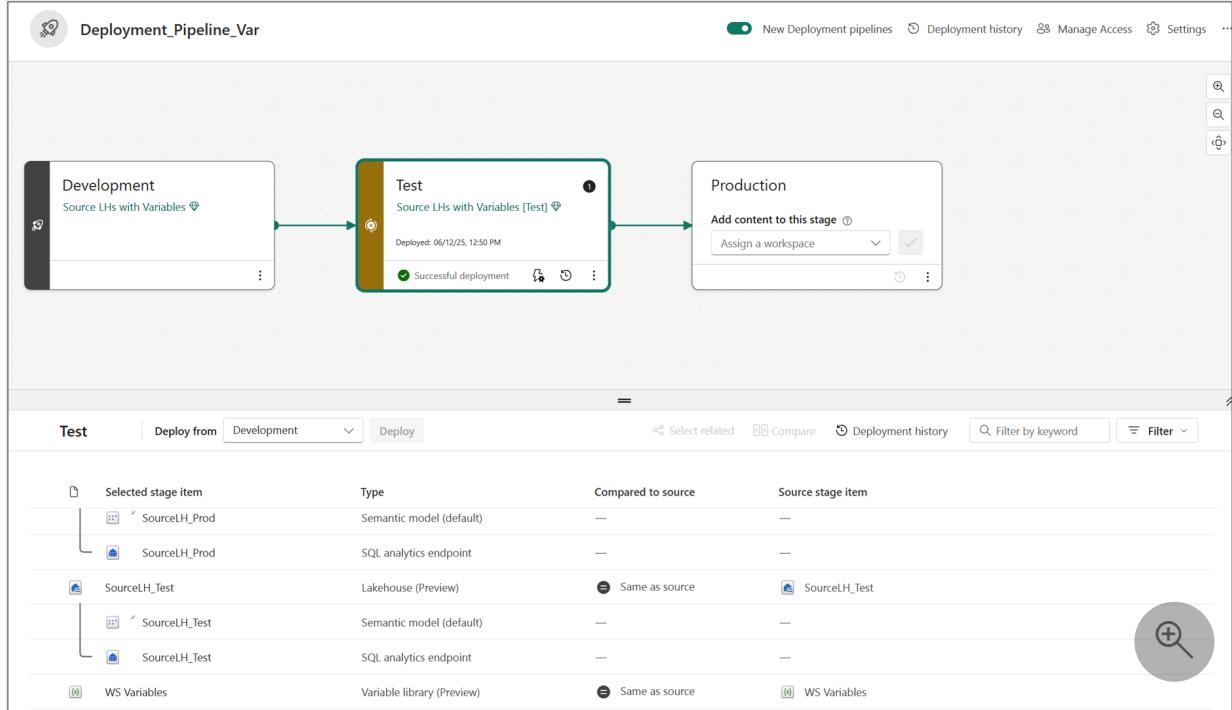
- Select the checkbox at the top to select all items. Then clear the checkbox for the **SourceLH_Dev** lakehouse.
- Select the **Deploy** button. Select **Deploy** again. The **Test** stage should now be populated.



6. For the **Production** stage:

- Select the checkbox at the top to select all items. Then clear the checkbox for the **SourceLH_Test** lakehouse.

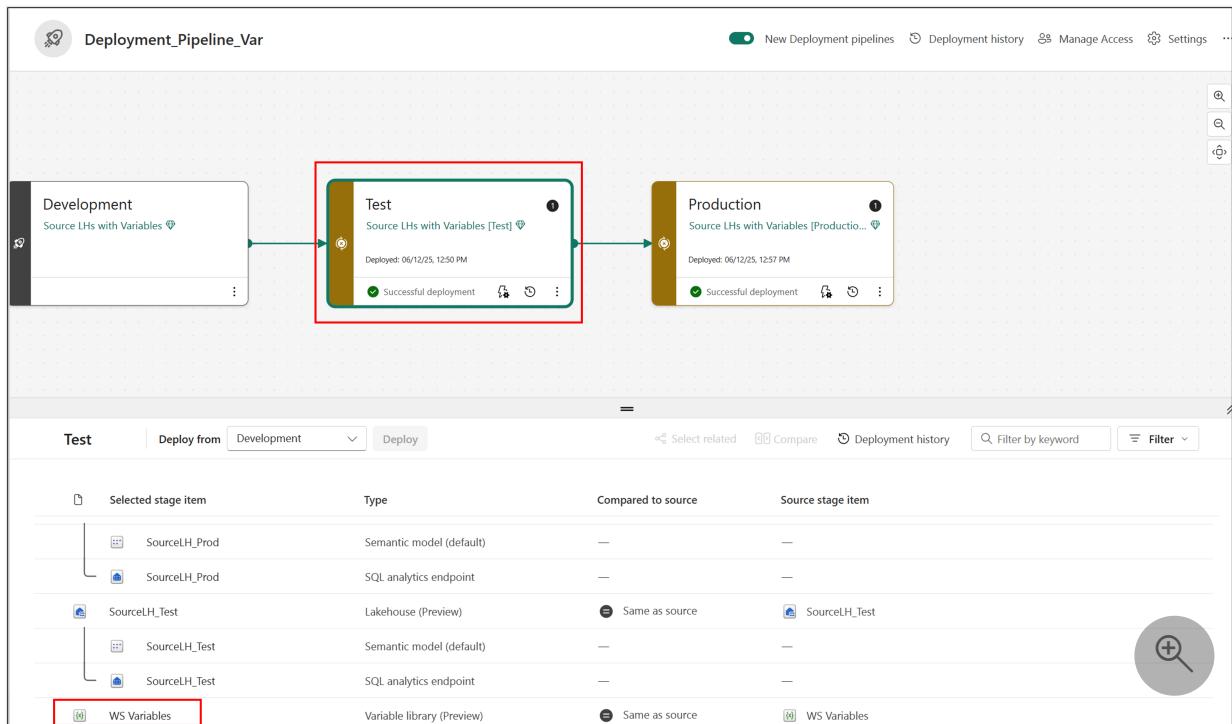
- b. Select the **Deploy** button. Select **Deploy** again. The **Production** stage should now be populated.



Set the variable library's active set for each stage

In these steps, you configure the active set for each stage in your deployment pipeline:

1. Configure the active set for the **Test** stage:
 - a. On the sidebar, select the **Deployment_Pipeline_Var** pipeline.
 - b. Select the **Test** stage.



c. Select WS Variables.

d. Select the ellipsis (...), and then select Set as active. Select the Set as Active button.

The screenshot shows the 'WS Variables' page. On the left, under 'Variables', there is a list of variables: Source_LH, Source_WSID, Destination_LH, Destination_WSID, SourceTable_Name, and DestinationTable_Name. Each variable has a checkbox, a note, and a type dropdown (String). To the right, under 'Default value set' (Active), there are several entries: 1f61c499-89cd-4df5-92c7-d110b..., 71bc08cb-a3dd-4d72-b101-1b3af..., 4fe228d3-a363-4b7f-a5d4-fae9d2..., ddfdf8621-3a7f-44ed-a44d-64ae48..., Processed, and DevCopiedData. On the right, under 'Alternative value sets', there are two sections: 'Test VS' and 'Prod VS'. The 'Test VS' section contains entries: 4fe228d3-a363-4b7f-a5d4-fae9d2... (highlighted with a red box), ddfdf8621-3a7f-44ed-a44d-64ae48..., c0f13027-9bf4-4e8c-8f57-ec5c18..., ddfdf8621-3a7f-44ed-a44d-64ae48..., DevCopiedData, and TestCopiedData. The 'Prod VS' section contains entries: 13027-9bf4-4e8c-8f57-ec5c18..., f8621-3a7f-44ed-a44d-64ae48..., 084e2ac5-386d-4c13-8f9a-d32fc5..., ddfdf8621-3a7f-44ed-a44d-64ae48..., TestCopiedData, and ProdCopiedData. A magnifying glass icon is on the right.

The active set is now configured.

The screenshot shows the 'WS Variables' page again. The 'Variables' section on the left is identical to the previous screenshot. On the right, both the 'Test VS' and 'Prod VS' sections are highlighted with green boxes. The 'Test VS' section contains the same entries as before. The 'Prod VS' section contains entries: c0f13027-9bf4-4e8c-8f57-ec5c18..., ddfdf8621-3a7f-44ed-a44d-64ae48..., 084e2ac5-386d-4c13-8f9a-d32fc5..., ddfdf8621-3a7f-44ed-a44d-64ae48..., TestCopiedData, and ProdCopiedData. A magnifying glass icon is on the right.

e. Select Save > Agree.

2. Configure the active set for the **Prod** stage:

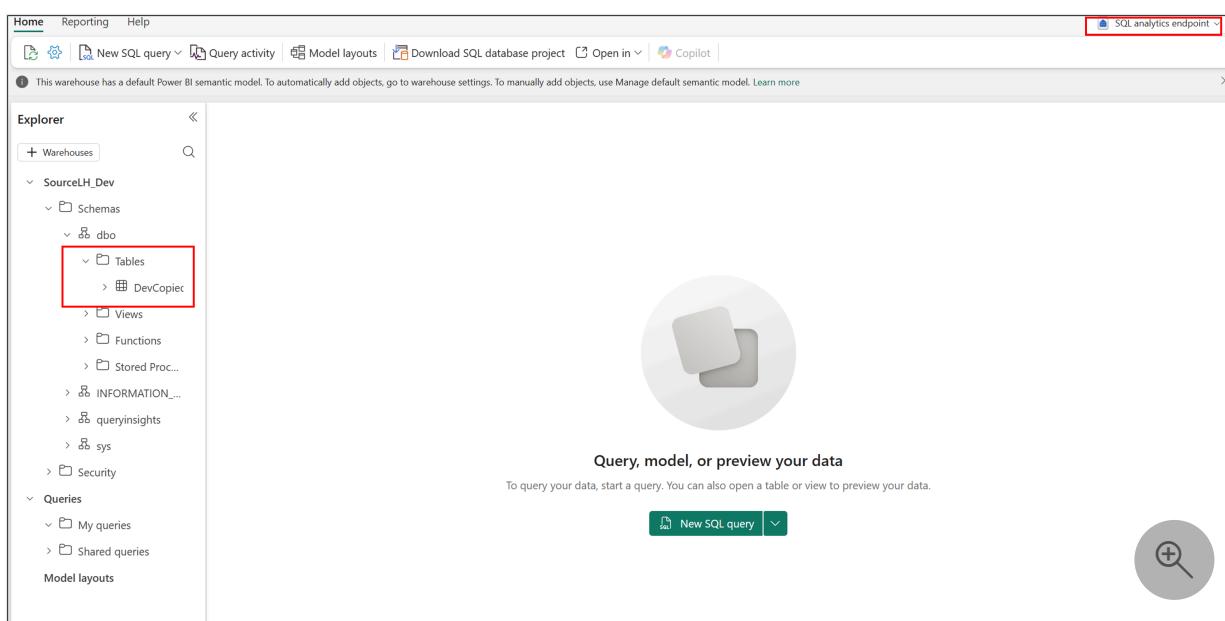
- a. On the sidebar, select the **Deployment_Pipeline_Var** pipeline.
- b. Select the **Prod** stage.
- c. Select **WS Variables**.
- d. Select the ellipsis (...), and then select **Set as active**. Select the **Set as Active** button.
- e. Select **Save > Agree**.

Verify and test the variable library

Now that you set up the variable library and configured all of the active sets for each stage of the deployment pipeline, you can verify them:

1. Check the **SourceLHs_Dev** lakehouse:

- a. In the **Source LHs with Variables** workspace, select the **SourceLHs_Dev** lakehouse.
- b. Change the connection from **Lakehouse** to **SQL analytics endpoint**.
- c. In the explorer, expand **Schemas > dbo > Tables**.
- d. Confirm that the **DevCopiedData** table appears.



2. Switch to the **SourceLHs_Test** lakehouse and repeat the preceding steps.

The **TestCopiedData** table shouldn't appear because you haven't run the pipeline yet with the **Test VS** active set.

3. Switch to the **SourceLHs_Prod** lakehouse and repeat the preceding steps.

The **ProdCopiedData** table shouldn't appear because you haven't run the pipeline yet with the **Prod VS** active set.

4. Check the **Test** stage of the **Deployment_Pipeline_Var** pipeline:

- Switch to the **Deployment_Pipeline_Var** pipeline and select the **Test** stage.
- Select the **Pipeline_Deploy** pipeline.
- Select **Run**. This process should finish successfully.

5. Check the **SourceLHs_Test** lakehouse again:

- Switch to the **SourceLHs_Test** lakehouse.
- Change the connection from **Lakehouse** to **SQL analytics endpoint**.
- In the explorer, expand **Schemas > dbo > Tables**.
- Confirm that the **TestCopiedData** table appears.

#	countryOrRegion	holidayName	normalizeHolidayName	isPaidTimeOff	countryRegionCode	date
1	United Kingdom	Silver Jubilee of Elizabeth II	Silver Jubilee of Elizabeth II	0	GB	1977-06-07 00:00:00.000000
2	United Kingdom	Wedding of Charles and Diana	Wedding of Charles and Diana	0	GB	1981-07-29 00:00:00.000000
3	United Kingdom	Millennium Celebrations	Millennium Celebrations	0	GB	1999-12-31 00:00:00.000000
4	United Kingdom	Golden Jubilee of Elizabeth II	Golden Jubilee of Elizabeth II	0	GB	2002-06-03 00:00:00.000000
5	United Kingdom	Wedding of William and Catherine	Wedding of William and Catherine	0	GB	2011-04-29 00:00:00.000000
6	United Kingdom	Diamond Jubilee of Elizabeth II	Diamond Jubilee of Elizabeth II	0	GB	2012-06-05 00:00:00.000000
7	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	1978-01-02 00:00:00.000000
8	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	1984-01-02 00:00:00.000000
9	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	1988-01-02 00:00:00.000000
10	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	1995-01-02 00:00:00.000000
11	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2006-01-02 00:00:00.000000
12	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2012-01-02 00:00:00.000000
13	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2017-01-02 00:00:00.000000
14	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2023-01-02 00:00:00.000000
15	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2034-01-02 00:00:00.000000
16	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2040-01-02 00:00:00.000000
17	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2045-01-02 00:00:00.000000
18	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2051-01-02 00:00:00.000000
19	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2062-01-02 00:00:00.000000
20	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2068-01-02 00:00:00.000000
21	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2073-01-02 00:00:00.000000
22	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2079-01-02 00:00:00.000000
23	United Kingdom	New Year Holiday [Scotland], New ...	New Year Holiday , New Year's Day	1	GB	2090-01-02 00:00:00.000000

6. Check the **Production** stage of the **Deployment_Pipeline_Var** pipeline:

- Switch to the **Deployment_Pipeline_Var** pipeline and select the **Production** stage.
- Select the **Pipeline_Deploy** pipeline.
- Select **Run**. This process should finish successfully.

7. Check the **SourceLHs_Prod** lakehouse again:

- Switch to the **SourceLHs_Prod** lakehouse.
- Change the connection from **Lakehouse** to **SQL analytics endpoint**.

c. In the explorer, expand **Schemas > dbo > Tables**.

d. Confirm that the **ProdCopiedData** table appears.

Customize the variable values in Git (optional)

To see how the variable library is [represented in Git](#), or to edit the variables from a Git repository:

1. In the workspace, select **Source control** and [connect the workspace to a Git repository](#).

2. On the **Source control** pane, select **Commit** to push the workspace content to the Git repository.

The Git repo has a folder for each item in the workspace. A folder called **WS variables.VariableLibrary** represents the variable library item. For more information about the contents of this folder, see [Variable library CI/CD](#).

3. Compare the **ProdVS.json** and **TestVS.json** files in the **valueSets** folder. Confirm that the **overrides** variable is set to the different values. You can edit these values directly in the UI or by editing this file in Git and updating it to the workspace.

JSON

```
{  
  "$schema": "https://developer.microsoft.com/json-  
schemas/fabric/item/VariablesLibrary/definition/valueSets/1.0.0/schema.json",  
  "valueSetName": "Test VS",  
  "overrides": [  
    {  
      "name": "Source_LH",  
      "value": "4fe228d3-a363-4b7f-a5d4-fae9d2abca43"  
    },  
    {  
      "name": "DestinationTableName",  
      "value": "TestCopiedData"  
    }  
  ]  
}
```

JSON

```
{  
  "$schema": "https://developer.microsoft.com/json-  
schemas/fabric/item/VariablesLibrary/definition/valueSets/1.0.0/schema.json",  
  "valueSetName": "Prod VS",  
  "overrides": [  
    {  
      "name": "Source_LH",  
      "value": "4fe228d3-a363-4b7f-a5d4-fae9d2abca43"  
    },  
    {  
      "name": "DestinationTableName",  
      "value": "ProdCopiedData"  
    }  
  ]  
}
```

```
        "name": "Source_LH",
        "value": "c0f13027-9bf4-4e8c-8f57-ec5c18c8656b"
    },
    {
        "name": "DestinationTableName",
        "value": "ProdCopiedData"
    }
]
```

Related content

- [Tutorial: Lifecycle management in Fabric](#)
- [Create and manage variable libraries](#)

Last updated on 12/15/2025

Troubleshoot variable libraries

This article provides solutions for common errors and problems that you might encounter when you work with Microsoft Fabric variable libraries.

Failure to manage a variable library

I can't create a variable library item

Description of the problem: I tried to create a variable library, but it failed.

Cause: The name of the variable library isn't valid.

Solution: Rename the variable library according to [naming conventions](#).

I can't save my variable library

Description of the problem: I tried to save my variable library, but it failed.

Cause: Reasons for this failure might include:

- The name of the variable or the value set isn't valid.
- The value isn't the correct type.
- The value is empty.
- The variable library size is greater than 1 MB.

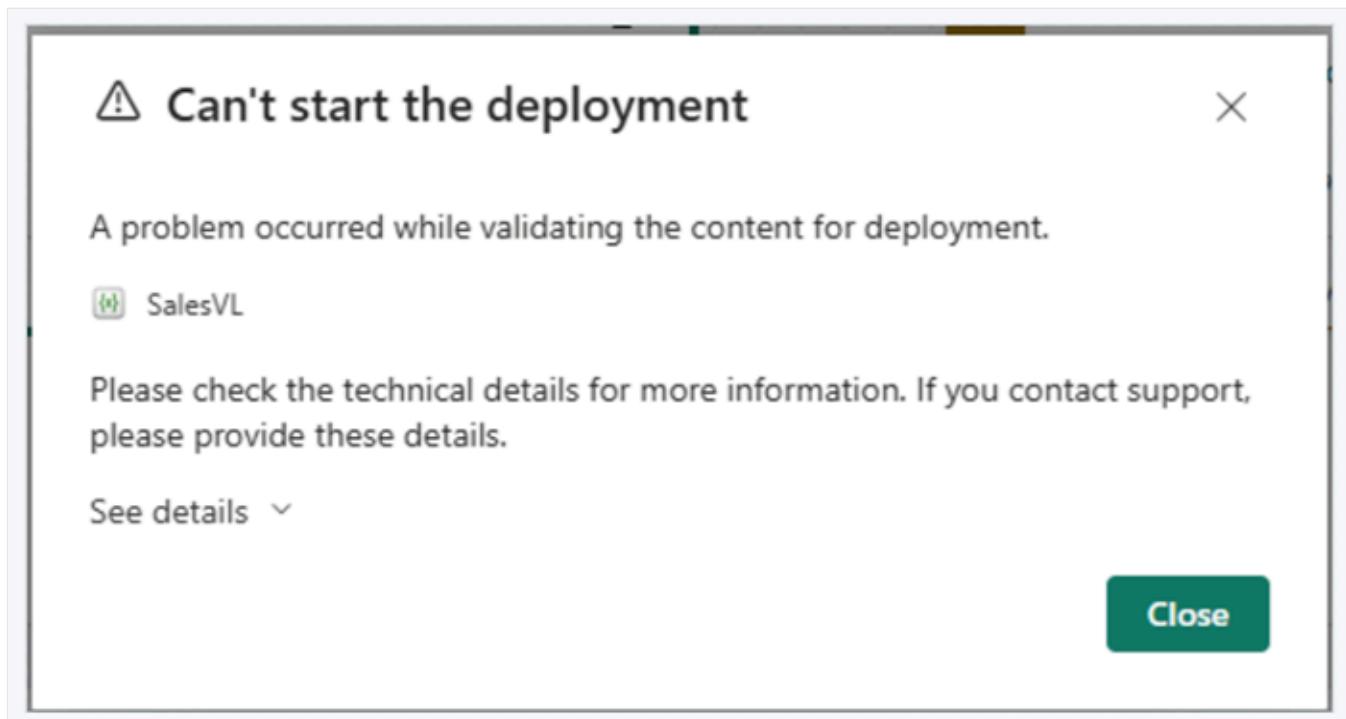
Solution: Depending on the cause, you can try the following solutions:

- Review the errors on the validation pane and fix them so that no required value is missing or mismatched with the variable type.
- Ensure that all variables and value sets are named according to the [naming conventions](#).
- Reduce the size of the variable library item to less than 1 MB by removing unused variables or by splitting the variable list into several variable libraries.

Deployment pipeline failure

I can't deploy my variable library in my deployment pipeline

Description of the problem: I tried to deploy the variable library, but I got a message that says "Can't start the deployment."



Cause: The active value set in the target stage is missing in the deployed variable library. A possible reason is that you removed or renamed the active value set in the source or target stage.

Solution: Change the active value set in the target stage, or rename the current value set to one that exists in the source.

Variable reference failure

I can't reference an added variable

Description of the problem: I can't find the variable that I want to refer to in the pipeline.

Cause: Reasons why a variable might not appear in the pipeline include:

- The variable wasn't saved.
- The variable was deleted or renamed.
- The variable was moved to another variable library.

Solution: Go back to the variable library and check the names of the existing variables. If the variable that you want exists but wasn't saved, save it. If it doesn't exist, create it.

After you find and fix the problem, remove the current reference and replace it with the correct name in the pipeline.

Active value set not changed

Description of the problem: I changed the active value set in the variable library, but I still get the value from the previous active value set.

Cause: You might have forgotten to save your changes in the variable library.

Solution: Go back to the variable library, ensure that the correct value set is active, and save.

Related content

- [Create and manage variable libraries](#)
-

Last updated on 12/15/2025

Best practices for lifecycle management in Fabric

This article provides guidance for data & analytics creators who are managing their content throughout its lifecycle in Microsoft Fabric. The article focuses on the use of [Git integration](#) for source control and [deployment pipelines](#) as a release tool. For a general guidance on Enterprise content publishing, [Enterprise content publishing](#).

The article is divided into four sections:

- [Content preparation](#) - Prepare your content for lifecycle management.
- [Development](#) - Learn about the best ways of creating content in the deployment pipelines development stage.
- [Test](#) - Understand how to use a deployment pipelines test stage to test your environment.
- [Production](#) - Utilize a deployment pipelines production stage to make your content available for consumption.

Best practices for content preparation

To best prepare your content for ongoing management throughout its lifecycle, review the information in this section before you:

- Release content to production.
- Start using a deployment pipeline for a specific workspace.

Separate development between teams

Different teams in the org usually have different expertise, ownership, and methods of work, even when working on the same project. It's important to set boundaries while giving each team their independence to work as they like. Consider having separate workspaces for different teams. With separate workspaces, each team can have different permissions, work with different source control repos, and ship content to production in a different cadence. Most items can connect and use data across workspaces, so it doesn't block collaboration on the same data and project.

Plan your permission model

Both Git integration and deployment pipelines require different permissions than just the workspace permissions. Read about the permission requirements for [Git integration](#) and [deployment pipelines](#).

To implement a secure and easy workflow, plan who gets access to each part of the environments being used, both the Git repository and the dev/test/prod stages in a pipeline. Some of the considerations to take into account are:

- Who should have access to the source code in the Git repository?
- Which operations should users with pipeline access be able to perform in each stage?
- Who's reviewing content in the test stage?
- Should the test stage reviewers have access to the pipeline?
- Who should oversee deployment to the production stage?
- Which workspace are you assigning to a pipeline, or connecting to git?
- Which branch are you connecting the workspace to? What's the policy defined for that branch?
- Is the workspace shared by multiple team members? Should they make changes directly in the workspace, or only through Pull requests?
- Which stage are you assigning your workspace to?
- Do you need to make changes to the permissions of the workspace you're assigning?

Connect different stages to different databases

A production database should always be stable and available. It's best not to overload it with queries generated by BI creators for their development or test semantic models. Build separate databases for development and testing in order to protect production data and not overload the development database with the entire volume of production data.

Use parameters for configurations that will change between stages

Whenever possible, add [parameters](#) to any definition that might change between dev/test/prod stages. Using parameters helps you change the definitions easily when you move your changes to production.

Parameters have different uses, such as defining connections to data sources, or to internal

items in Fabric. They can also be used to make changes to queries, filters, and the text displayed to users.

In deployment pipelines, you can configure parameter rules to set different values for each deployment stage.

Best practices for deployment pipelines development stage

This section provides guidance for working with the deployment pipelines and using it for your development stage.

Back up your work into a Git repository

With Git integration, any developer can back up their work by committing it into Git. To back up your work properly in Fabric, here are some basic rules:

- Make sure you have an isolated environment to work in, so others don't override your work before it gets committed. This means working in a Desktop tool (such as [VS Code](#), [Power BI Desktop](#) or others), or in a separate workspace that other users can't access.
- Commit to a branch that you created and no other developer is using. If you're using a workspace as an authoring environment, read about [working with branches](#).
- Commit together changes that must be deployed together. This advice applies for a single item, or multiple items that are related to the same change. Committing all related changes together can help you later when deploying to other stages, creating pull requests, or reverting changes back.
- Big commits might hit a max commit size limit. Be mindful of the number of items you commit together, or the general size of an item. For example, reports can grow large when adding large images. It's bad practice to store large-size items in source control systems, even if it works. Consider ways to reduce the size of your items if they have lots of static resources, like images.

Rolling back changes

After you back up your work, there might be cases where you want to revert to a previous version and restore it in the workspace. There are a few ways to revert to a previous version:

- **Undo button:** The *Undo* operation is an easy and fast way to revert the immediate changes you made, as long as they aren't committed yet. You can also undo each item

separately. Read more about the [undo](#) operation.

- **Reverting to older commits:** There's no direct way to go back to a previous commit in the UI. The best option is to promote an older commit to be the HEAD using [git revert](#) or [git reset](#). Doing this shows that there's an update in the source control pane, and you can update the workspace with that new commit.

Since data isn't stored in Git, keep in mind that reverting a data item to an older version might break the existing data and could possibly require you to drop the data or the operation might fail. Check this in advance before reverting changes back.

Working with a 'private' workspace

When you want to work in isolation, use a separate workspace as an isolated environment.

Read more about isolating your work environment in [working with branches](#). For an optimal workflow for you and the team, consider the following points:

- **Setting up the workspace:** Before you start, make sure you can create a new workspace (if you don't already have one), that you can assign it to a [Fabric capacity](#), and that you have access to data to work in your workspace.
- **Creating a new branch:** Create a new branch from the *main* branch, so you have the most up-to-date version of your content. Also make sure you connect to the correct folder in the branch, so you can pull the right content into the workspace.
- **Small, frequent changes:** It's a Git best practice to make small incremental changes that are easy to merge and less likely to get into conflicts. If that's not possible, make sure to update your branch from main so you can resolve conflicts on your own first.
- **Configuration changes:** If necessary, change the configurations in your workspace to help you work more productively. Some changes can include connection between items, or to different data sources or changes to parameters on a given item. Just remember that anything you commit becomes part of the commit and can accidentally be merged into the main branch.

Use Client tools to edit your work

For items and tools that support it, it might be easier to work with client tools for authoring, such as [Power BI Desktop](#) for semantic models and reports, [VS Code](#) for Notebooks etc. These tools can be your local development environment. After you complete your work, push the changes into the remote repo, and sync the workspace to upload the changes. Just make sure you're working with the [supported structure](#) of the item you're authoring. If you're not

sure, first clone a repo with content already synced to a workspace, then start authoring from there, where the structure is already in place.

Managing workspaces and branches

Since a workspace can only be connected to a single branch at a time, it's recommended to treat this as a 1:1 mapping. However, to reduce the amount of workspace it entails, consider these options:

- If a developer set up a private workspace with all required configurations, they can continue to use that workspace for any future branch they create. When a sprint is over, your changes are merged and you're starting a fresh new task, just switch the connection to a new branch on the same workspace. You can also do this if you suddenly need to fix a bug in the middle of a sprint. Think of it as a working directory on the web.
- Developers using a client tool (such as VS Code, Power BI Desktop, or others), don't necessarily need a workspace. They can create branches and commit changes to that branch locally, push those to the remote repo and create a pull request to the main branch, all without a workspace. A workspace is needed only as a testing environment to check that everything works in a real-life scenario. It's up to you to decide when that should happen.

Duplicate an item in a Git repository

To duplicate an item in a Git repository:

1. Copy the entire item directory.
2. Change the logicalId to something unique for that connected workspace.
3. Change the display name to differentiate it from the original item and to avoid duplicate display name error.
4. If necessary, update the logicalId and/or display names in any dependencies.

Best practices for deployment pipelines test stage

This section provides guidance for working with a deployment pipelines test stage.

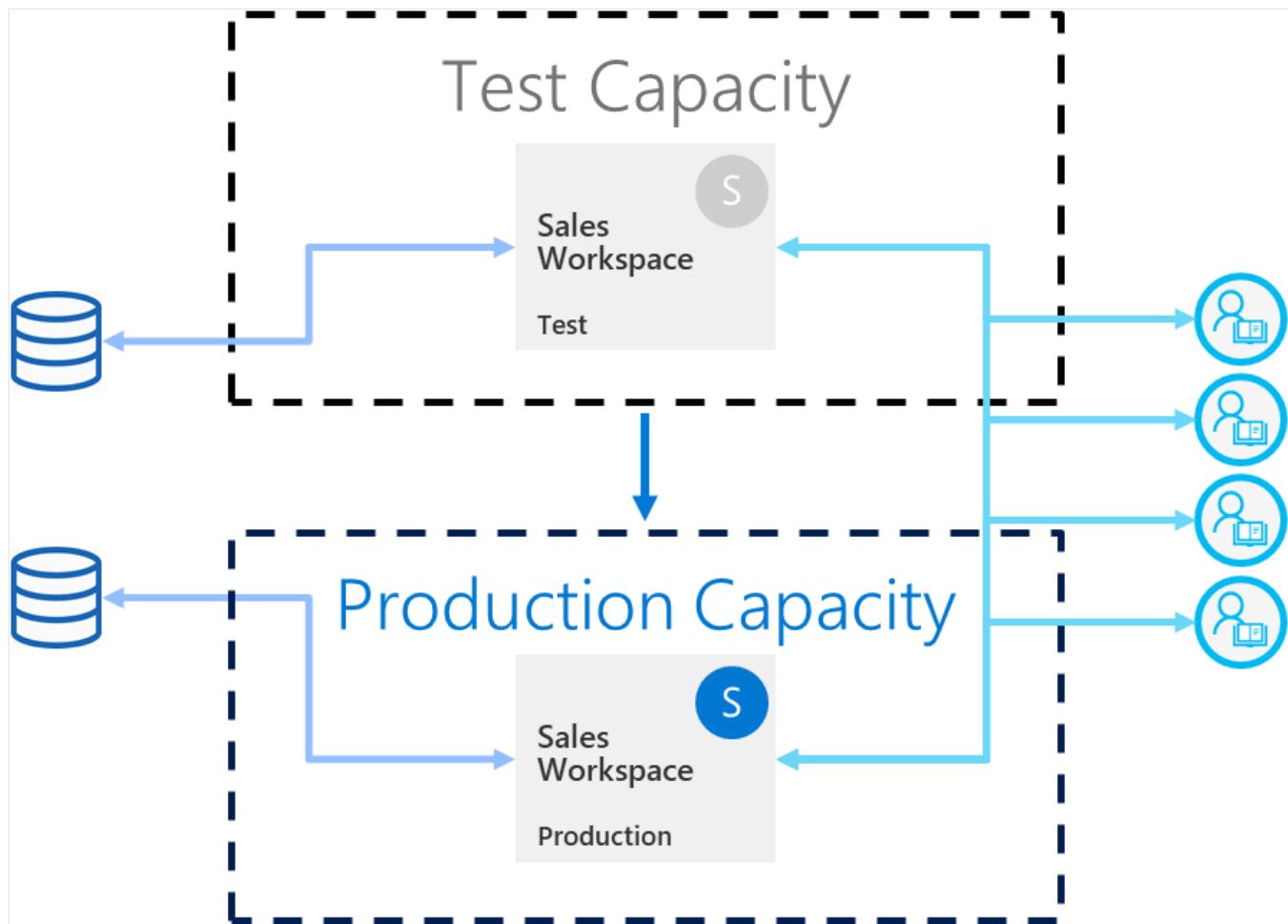
Simulate your production environment

It's important to see how your proposed change will impact the production stage. A deployment pipelines test stage allows you to simulate a real production environment for testing purposes. Alternatively, you can simulate this by connecting Git to another workspace.

Make sure that these three factors are addressed in your test environment:

- Data volume
- Usage volume
- A similar capacity as in production

When testing, you can use the same capacity as the production stage. However, using the same capacity can make production unstable during load testing. To avoid unstable production, test using a different capacity similar in resources to the production capacity. To avoid extra costs, use a capacity where you can pay only for the testing time.



Use deployment rules with a real-life data source

If you're using the test stage to simulate real life data usage, it's best to separate the development and test data sources. The development database should be relatively small, and the test database should be as similar as possible to the production database. Use [data source rules](#) to switch data sources in the test stage or parameterize the connection if not working through deployment pipelines.

Check related items

Changes you make can also affect the dependent items. During testing, verify that your changes don't affect or break the performance of existing items, which can be dependent on the updated ones.

You can easily find the related items by using [impact analysis](#).

Updating data items

Data items are items that store data. The item's definition in Git defines how the data is stored. When updating an item in the workspace, we're importing its definition into the workspace and applying it on the existing data. The operation of updating data items is the same for Git and deployment pipelines.

As different items have different capabilities when it comes to retaining data when changes to the definition are applied, be mindful when applying the changes. Some practices that can help you apply the changes in the safest way:

- Know in advance what the changes are and what their impact might be on the existing data. Use commit messages to describe the changes made.
- To see how that item handles the change with test data, upload the changes first to a dev or test environment.
- If everything goes well, it's recommended to also check it on a staging environment, with real-life data (or as close to it as possible), to minimize the unexpected behaviors in production.
- Consider the best timing when updating the Prod environment to minimize the damage that any errors might cause to your business users who consume the data.
- After deployment, post-deployment tests in Prod to verify that everything is working as expected.
- Some changes will always be considered *breaking changes*. Hopefully, the preceding steps will help you track them before production. Build a plan for how to apply the changes in Prod and recover the data to get back to normal state and minimize downtime for business users.

Test your app

If you're distributing content to your customers through an app, review the app's new version *before* it's in production. Since each deployment pipeline stage has its own workspace, you can

easily publish and update apps for development and test stages. Publishing and updating apps allows you to test the app from an end user's point of view.

Important

The deployment process doesn't include updating the app content or settings. To apply changes to content or settings, manually update the app in the required pipeline stage.

Best practices for deployment pipelines production stage

This section provides guidance to the deployment pipelines production stage.

Manage who can deploy to production

Because deploying to production should be handled carefully, it's good practice to let only specific people manage this sensitive operation. However, you probably want all BI creators for a specific workspace to have access to the pipeline. Use production [workspace permissions](#) to manage access permissions. Other users can have a production workspace *viewer* role to see content in the workspace but not make changes from Git or deployment pipelines.

In addition, limit access to the repo or pipeline by only enabling permissions to users that are part of the content creation process.

Set rules to ensure production stage availability

[Deployment rules](#) are a powerful way to ensure the data in production is always connected and available to users. With deployment rules applied, deployments can run while you have the assurance that customers can see the relevant information without disturbance.

Make sure that you set production deployment rules for data sources and parameters defined in the semantic model.

Update the production app

Deployment in a pipeline through the UI updates the workspace content. To update the associated app, use the [deployment pipelines API](#). It's not possible to update the app through the UI. If you use an app for content distribution, don't forget to update the app after deploying to production so that end users are immediately able to use the latest version.

Deploying into production using Git branches

As the repo serves as the ‘single-source-of-truth’, some teams might want to deploy updates into different stages directly from Git. This is possible with Git integration, with a few considerations:

- We recommend using release branches. You need to continuously change the connection of workspace to the new release branches before every deployment.
- If your build or release pipeline requires you to change the source code, or run scripts in a build environment before deployment to the workspace, then connecting the workspace to Git won't help you.
- After deploying to each stage, make sure to change all the configuration specific to that stage.

Quick fixes to content

Sometimes there are issues in production that require a quick fix. Deploying a fix without testing it first is bad practice. Therefore, always implement the fix in the development stage and push it to the rest of the deployment pipeline stages. Deploying to the development stage allows you to check that the fix works before deploying it to production. Deploying across the pipeline takes only a few minutes.

If you're using deployment from Git, we recommend following the practices described in [Adopt a Git branching strategy](#).

Related content

- [End to end lifecycle management tutorial](#)
- [Get started with Git integration](#)
- [Get started with deployment pipelines](#)

Network security for continuous integration/continuous deployment

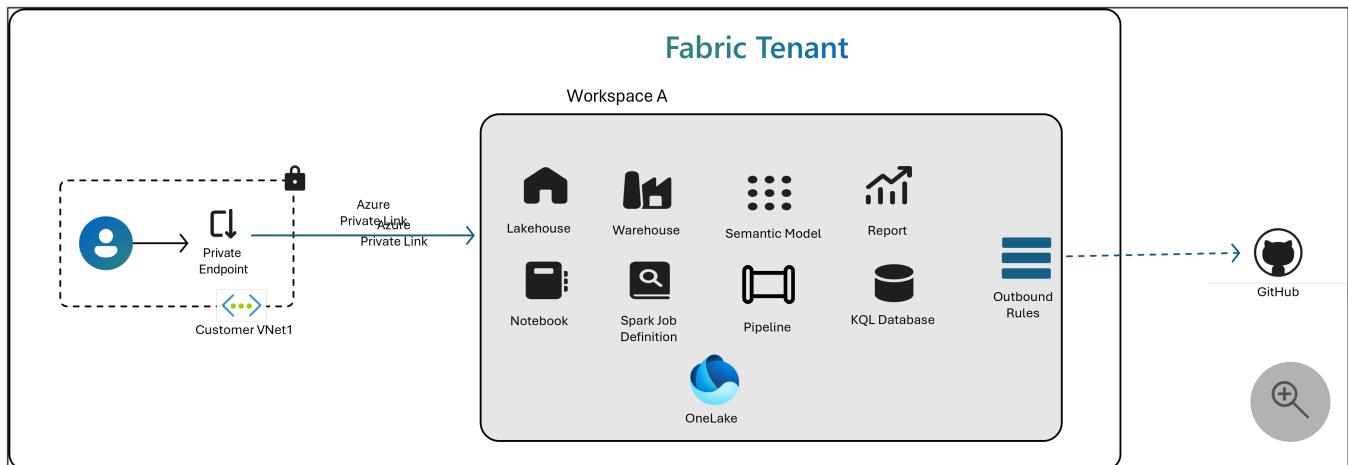
Microsoft Fabric is a software as a service (SaaS) platform that lets users get, create, share, and visualize data. As a SaaS service, Fabric offers a complete security package for the entire platform. For more information, see [Network Security](#)

ⓘ Important

Deployment pipelines are currently not supported with workspace inbound and outbound access protection.

Workspace level security

Workspaces represent the primary security boundary for data stored in OneLake. Each workspace represents a single domain or project area where teams can collaborate on data. Workspace-level security in Microsoft Fabric provides granular control over data access and network connectivity by allowing administrators to configure both inbound and outbound protections for individual workspaces.



Workspace level security is composed of two main features.

- **Workspace inbound access protection** - is a network security feature that uses [private links](#) to ensure that connections to a workspace are from secure and approved networks. Private links enable secure connectivity to Fabric by restricting access to your Fabric tenant or workspace from an Azure virtual network (VNet), and blocking all public access. For more information, see [Manage admin access to workspace inbound access protection settings](#)

- **Workspace Outbound Access Protection (OAP)** - allows administrators to control and restrict outbound connections from workspace artifacts to external resources. For outbound security, Fabric supports workspace outbound access protection. This network security feature ensures that connections outside the workspace go through a secure connection between Fabric and a virtual network. It prevents items from establishing unsecure connections to sources outside the workspace boundary unless allowed by the workspace admins. This granular control makes it possible to restrict outbound connectivity for some workspaces while allowing the rest of the workspaces to remain open. For more information, see [Workspace outbound access protection \(preview\)](#)

Git integration and network security

Git integration in Fabric lets a workspace sync its content (like notebooks, dataflows, Power BI reports, etc.) with an external Git repository (GitHub or Azure DevOps). Because the workspace must pull from or push to a Git service outside of Fabric, it involves outbound communication.

Workspace inbound access and Git integration

When Private Link is enabled for a workspace, users must connect through a designated virtual network (VNet), effectively isolating the workspace from public internet exposure.

This restriction directly impacts Git integration: users attempting to access Git features (such as syncing or committing changes) must do so from within the approved VNet. If a user tries to open the Git pane or perform Git operations from an unapproved network, Fabric blocks access to the workspace UI entirely, including Git functionality. This restriction also applies to the Git APIs. This enforcement ensures that Git-related actions—like connecting to a repository or branching out—are only performed in secure, controlled environments, reducing the risk of data leakage through source control channels.

Git integration with inbound access protection is enabled by default. There's no toggle to disable. Branching out is blocked.

Workspace outbound access and Git integration

By default, Workspace OAP will completely block Git integration, because contacting an external Git endpoint would violate the "no outbound" rule. To resolve this restriction, Fabric introduces an admin-controlled consent setting for Git.

How Git integration works with OAP

Each workspace with OAP enabled has an explicit toggle (a checkbox in the workspace's network settings) labeled to allow Git integration for that workspace. Initially, when OAP is turned on, this checkbox is off by default – meaning no Git connectivity is allowed. In that state, if a user opens the workspace's Git panel in Fabric, they'll see the Git features disabled (greyed out) with an explanation that "Outbound access is restricted".

Similarly, any attempt to call Git APIs (e.g. via automation or PowerShell) for that workspace fails with an error as long as Git is disallowed. This protection ensures that, by default, a secured workspace can't quietly sync its content to an external repository without awareness.

Outbound access protection

Block outbound public access  On

Block all outbound connections from workspace and only allow connections through private end points and connection rules. This feature is only available for limited items. Creation of other items will be disabled once this feature is turned on. Please wait for 15 mins for the setting change to take effect. [Learn more](#) 

Allow Git Integration  On

Use all Git features, like commit and sync, for the repository connected to this workspace. If this is off, Git integration will be blocked. 

To enable Git integration, an administrator can go to the workspace's **Outbound security settings** and clicks the **Allow Git integration** toggle. (This box can only be checked after OAP itself is enabled; it's a sub-option under outbound settings.) Checking **Allow Git integration** is effectively the admin giving consent that this workspace is permitted to communicate with Git.

 **Note**

The Git integration consent is per workspace.

Once enabled, Fabric immediately lifts the restrictions on Git for that workspace: the Git UI becomes active and all operations – connecting a repo, syncing (pull/push), committing changes, and branch management – are now allowed for users in that workspace.

 **Note**

When OAP isn't enabled, the toggle won't impact git integration and can't be turned on or off.

The following table summarizes how git integration works with OAP.

[\[+\] Expand table](#)

OAP state	Allow Git integration toggle	Git integration state
Enabled	Off	Git integration won't work
Enabled	On	Git integration will work
Disabled	Greyed out	Git integration not impacted

Enable Git Integration

To use workspace outbound access protection and enable git integration, do the following:

1. Sign-in to the fabric portal
2. Navigate to your workspace
3. In the top right, select workspace settings.
4. On the right, click **Outbound networking**.
5. Under **Outbound access protection (preview)**, make sure **Allow Git Integration** is toggled to on.

Branch out considerations

The **Branch Out** feature creates a new workspace from the current Git branch, or links to an existing workspace, and is a special case under OAP. When Git integration is allowed via admin consent, branching out is also allowed. Fabric provides a clear warning in the branch-out dialog if you attempt to branch into a workspace that doesn't have OAP enabled.

For example, if you're branching out from a locked-down dev workspace to create a new test workspace, a warning will state that the new workspace won't automatically have outbound protection.

Note

Workspace settings aren't copied over to other workspaces through git, or directly. This applies to branching out or copied workspaces. OAP and Git integration need to be enabled after the new workspace is created.

New workspaces start with OAP off by default and the administrator should manually enable OAP on the new workspace after it's created via branch-out to maintain the same security level. If branching out to an existing workspace, the warning will appear if that target workspace isn't OAP-protected. This is to prevent an unaware user from pushing content into an environment that undermines security.

Removing Git integration from OAP

Once Git is allowed, the workspace will operate normally with respect to source control. If at any point an administrator decides to disable Git integration, they can click the **Allow Git integration** toggle. Fabric will then immediately cut off the Git connectivity for that workspace. Any subsequent Git operations (pull, push, etc.) fails, and the UI will revert to a disabled state requiring re-approval.

To prevent accidental disruption, Fabric provides a confirmation/warning to the administrator when turning off Git access, explaining that all Git sync for that workspace will stop. It's worth noting that disabling Git doesn't delete the repository or any history – it severs the connection from the workspace side.

REST API support

Administrators can use REST APIs to programmatically query the network settings of workspaces. These queries can be done to indicate if outbound protection is enabled or if you want to set the outbound policy. These allow scripting of audits – you could retrieve all workspaces and check which ones have gitAllowed: true under OAP. Using such APIs, a security team could, for instance, nightly confirm that no additional workspaces have Git allowed without approval. Microsoft has introduced the following endpoints to get or set the Git outbound policy for a workspace

The GET /workspaces/{workspaceId}/gitOutboundPolicy API allows administrators or automation systems to retrieve the current outbound Git policy for a specific workspace. This action is particularly useful for auditing and compliance purposes, as it confirms whether Git operations (such as repo sync, commit, or branch-out) are permitted under the workspace's outbound access protection settings. Checking this policy, allows users, to ensure that only explicitly approved workspaces are allowed to interact with external Git repositories, helping prevent unintended data exfiltration.

The SET /workspaces/{workspaceId}/gitOutboundPolicy API enables administrators to programmatically configure the Git outbound policy for a workspace. This configuration includes toggling the consent that allows Git operations even when DEP is enabled. Automating this configuration via API is beneficial for CI/CD workflows, allowing secure workspaces to be onboarded into Git-based development pipelines without manual

intervention. It also supports infrastructure-as-code practices, where network and integration policies are versioned and deployed alongside workspace configurations.

Auditing and logs

The Fabric platform logs events whenever an operation is blocked due to network security. A high volume of such errors might indicate either misconfiguration (someone forgot to enable a needed setting) or a potential attempt to bypass security. For more information, see [Track user activities in Microsoft Fabric](#)

Limitations and considerations

The following is information you need to keep in mind when using OAP and Git integration.

- Not all items support inbound and outbound access protection. Syncing unsupported items into the workspace from git integration will fail. For a list of supported items, see [Private link supported items](#) and [Outbound access protection supported items](#).
- Deployment pipelines are currently not supported with workspace inbound access protection.
- If the workspace is part of Deployment Pipelines, workspace admins can't enable outbound access protection because Deployment Pipelines are unsupported. Similarly, if outbound access protection is enabled, the workspace can't be added to Deployment Pipelines.

For more information, see [OAP and workspace considerations](#)

Related content

- [Git integration](#)
- [Manage admin access to workspace inbound access protection settings](#)
- [Workspace outbound access protection](#)

Frequently asked questions about the Fabric lifecycle management tools

FAQ

This article provides answers to some of the most common questions about the Fabric lifecycle management tools.

General questions

What is lifecycle management in Microsoft Fabric?

Lifecycle management has two parts, integration and deployment. To understand what integration is in Fabric, refer to [the Git integration overview](#). To understand what deployment pipelines is in Fabric, refer to the [deployment pipelines overview](#).

What is Git integration?

For a short explanation of Git integration, see the [Git integration overview](#). A multiline or formatted answer to the question. Use any Markdown formatting you want, provided that you maintain the indent on the lines after the | characters.

What is deployment pipelines?

For a short explanation of deployment pipelines, see the [deployment pipelines overview](#).

Licensing questions

What licenses are needed to work with lifecycle management?

For information on licenses, see [Fabric licenses](#).

What type of capacity do I need?

All workspaces must be assigned to a Fabric license. However, you can use different capacity types for different workspaces.

For information on capacity types, see [Capacity and SKUs](#).

Note

- PPU, EM, and A SKUs only work with Power BI items. If you add other Fabric items to the workspace, you need a trial, P, or F SKU.
- When you create a workspace with a PPU, only other PPU users can able to access the workspace and consume its content.

Permissions

What is the deployment pipelines permissions model?

The deployment pipelines permissions model is described the [permissions](#) section.

What permissions do I need to configure deployment rules?

To configure deployment rules in deployment pipelines, you must be the owner of the semantic model.

Git integration questions

Can I connect to a repository that's in a different region than my workspace?

If the workspace capacity is in one geographic location while the Azure DevOps repo is in another location, the Fabric admin can decide whether to enable cross-geo exports. For more information, see [Users can export items to Git repositories in other geographical locations](#).

How do I get started with Git integration?

Get started with Git integration using the [get started instructions](#).

Why was my item removed from the workspace?

There could be several reasons why an item was removed from the workspace.

- If the item wasn't committed and you selected it in an *undo* action, the item is removed from the workspace.
- If the item was committed, it could get removed if you switch branches and the item doesn't exist in the new branch.

Deployment pipelines questions

What are some general deployment limitations to keep in mind?

The following considerations are important to keep in mind:

- [Deployment rule limitations](#)
- [Supported data sources for dataflow and semantic model rules](#)
- [Incremental refresh](#)
- [Automation](#)

How can I assign workspaces to all the stages in a pipeline?

You can either assign one workspace to your pipeline and deploy it across the pipeline, or assign a different workspace to each pipeline stage. For more information, see [assign a workspace to a deployment pipeline](#).

What can I do if I have a dataset with DirectQuery or Composite connectivity mode that uses variation or auto date/time tables?

Datasets that use DirectQuery or Composite connectivity mode and have variation or [auto date/time](#) tables aren't supported in deployment pipelines. If your deployment fails and you think it's because you have a dataset with a variation table, you can look for the [variations](#) property in your table's columns. You can use one of the following methods to edit your semantic model so that it works in deployment pipelines.

- In your dataset, instead of using DirectQuery or Composite mode, use [import](#) mode.
- Remove the [auto date/time](#) tables from your semantic model. If necessary, delete any remaining variations from all the columns in your tables. Deleting a variation can invalidate user authored measures, calculated columns, and calculated tables. Use this

method only if you understand how your semantic model works as it can result in data corruption in your visuals.

Why aren't some tiles displaying information after deployment?

When you pin a tile to a dashboard, if the tile relies on an unsupported item (any item not on [this list](#) is unsupported), or on an item that you don't have permissions to deploy, after deploying the dashboard the tile doesn't render. For example, if you create a tile from a report that relies on a semantic model you're not an admin on, when deploying the report you get an error warning. However, when deploying the dashboard with the tile, you don't get an error message, the deployment will succeed, but the tile won't display any information.

Paginated reports

Who's the owner of a deployed paginated report?

The owner of a deployed paginated report is the user who deployed the report. When you're deploying a paginated report for the first time, you become the owner of the report.

If you're deploying a paginated report to a stage that already contains a copy of that paginated report, you overwrite the previous report and become its owner, instead of the previous owner. In such cases, you need credentials to the underlying data source, so that the data can be used in the paginated report.

Where are my paginated report subreports?

Paginated report subreports are kept in the same folder that holds your paginated report. To avoid rendering problems, when you're using [selective copy](#) to copy a paginated report with subreports, select both the parent report and the subreports.

How do I create a deployment rule for a paginated report with a Fabric semantic model?

Paginated report rules can be created if you want to point the paginated report to the semantic model in the same stage. When creating a deployment rule for a paginated report, you need to select a database and a server.

If you're setting a deployment rule for a paginated report that doesn't have a Fabric semantic model, because the target data source is external, you need to specify both the server and the

database.

However, paginated reports that use a Fabric semantic model use an internal semantic model. In such cases, you can't rely on the data source name to identify the Fabric semantic model you're connecting to. The data source name doesn't change when you update it in the target stage, by creating a data source rule or by calling the [update datasource](#) API. When you set a deployment rule, you need to keep the database format and replace the semantic model object ID in the database field. As the semantic model is internal, the server stays the same.

- **Database** - The database format for a paginated report with a Fabric semantic model, is `sobe_wowvirtualserver-<dataset ID>`. For example, `sobe_wowvirtualserver-d51fd26e-9124-467f-919c-0c48a99a1d63`. Replace the `<dataset ID>` with your dataset's ID. You can get the dataset ID from the URL, by selecting the GUID that comes after `datasets/` and before the next forward slash.

[powerbi.com/groups/97341742-1a52-416b-a331-e6c2c78e7a4e/datasets/d51fd26e-9124-467f-919c-0c48a99a1d63/...](http://powerbi.com/groups/97341742-1a52-416b-a331-e6c2c78e7a4e/datasets/d51fd26e-9124-467f-919c-0c48a99a1d63/)

- **Server** - The server that hosts your database. Keep the existing server as is.

After deployment, can I download the RDL file of the paginated report?

After a deployment, if you download the RDL of the paginated report, it might not be updated with the latest version that you can see in Power BI service.

Dataflows

What happens to the incremental refresh configuration after deploying dataflows?

When you have a dataflow that contains semantic models that are configured with [incremental refresh](#), the refresh policy isn't copied or overwritten during deployment. After deploying a dataflow that includes a semantic model with incremental refresh to a stage that doesn't include this dataflow, if you have a refresh policy you'll need to reconfigure it in the target stage. If you're deploying a dataflow with incremental refresh to a stage where it already resides, the incremental refresh policy isn't copied. In such cases, if you wish to update the refresh policy in the target stage, you need to do it manually.

Datamarts

Where is my datamart's dataset?

Deployment pipelines don't display datasets that belong to datamarts in the pipeline stages. When you're deploying a datamart, its dataset is also deployed. You can view your datamart's dataset in the workspace of the stage it's in.

Related content

- [Get started with Git integration](#)
- [Get started with deployment pipelines](#)

Troubleshoot lifecycle management issues

Use this article to troubleshoot issues in the lifecycle management process.

To understand the considerations and limitations of various lifecycle management issues, review the links in the following table:

[+] [Expand table](#)

Topic	Git integration	Deployment pipelines
General limitations	general Git limitations	deployment pipelines limitations
Permissions needed	permissions	permissions
Workspace limitations	workspaces	workspaces
Supported Fabric items	supported items	supported items
Semantic model		Semantic model limitations

- [Git integration](#)
 - [Access](#)
 - [Connect](#)
 - [Commit](#)
 - [Update](#)
 - [Undo](#)
 - [Resolve errors](#)
- [Deployment pipelines](#)
 - [Paginated reports](#)
 - [Dataflows](#)
 - [Datamarts](#)
 - [Permissions](#)
 - [Rules](#)

Git integration

Access issues

I can't access my Azure DevOps repository

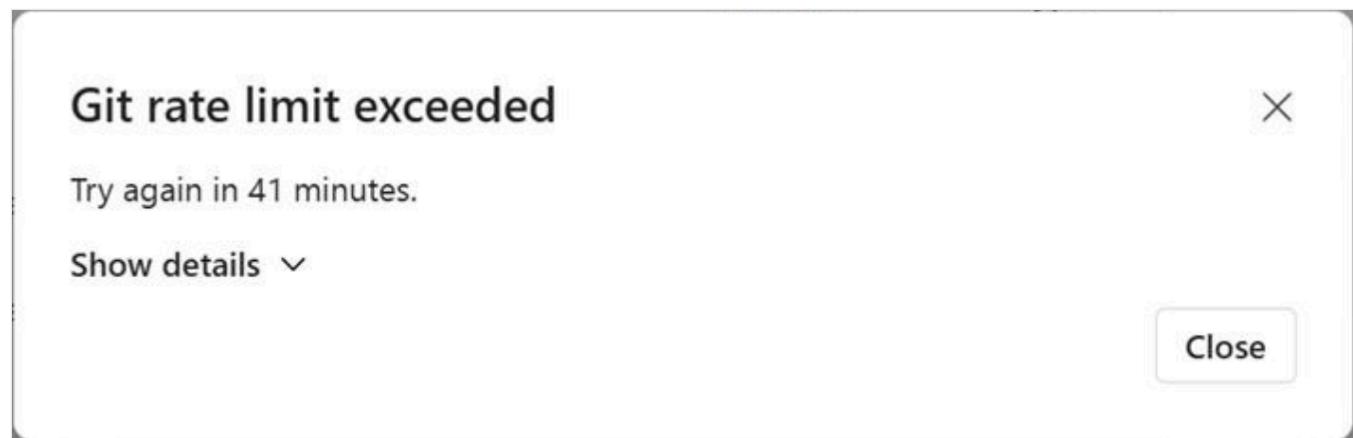
Description of problem: When I go to the Git integration tab, I get an error message and can't access Azure DevOps.

Cause: If the authentication method in Power BI is weaker than the authentication method in Azure DevOps, the functionalities between them doesn't work.

Workaround: The admin needs to align the authentication method in Power BI and Azure DevOps. The authentication policies for Microsoft Entra ID (formerly known as Azure Active Directory) are defined in [Manage authentication methods](#).

I exceeded the Git rate limit

Description of problem: When I try to update or commit to Git, I get an error message that says that I exceeded the Git rate limit.



Cause: Your git provider limits the number of Git actions you can perform in a given amount of time. The rate limit can be reached either by performing a large number of Git operations, or by executing operations that involve a large number of items. For information about GitHub rate limits, see [About primary rate limits](#). For Azure DevOps limits, see [Rate and usage limits](#).

Solution: Wait the amount of time specified in the error message, and then try again. If you continue to see this error, contact your Git provider for more information.

Connect issues

Connect failure: Unable to connect to repository

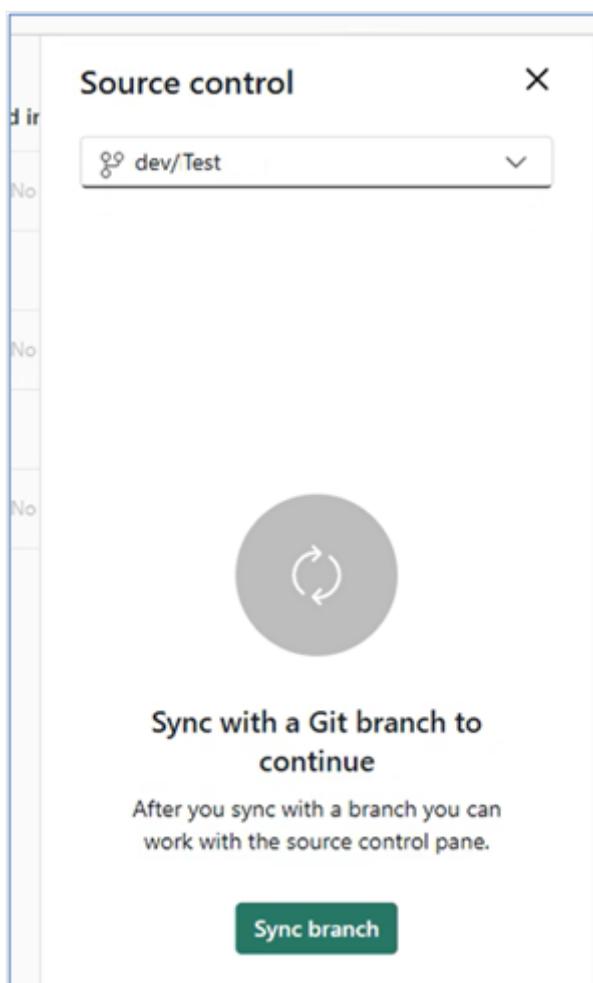
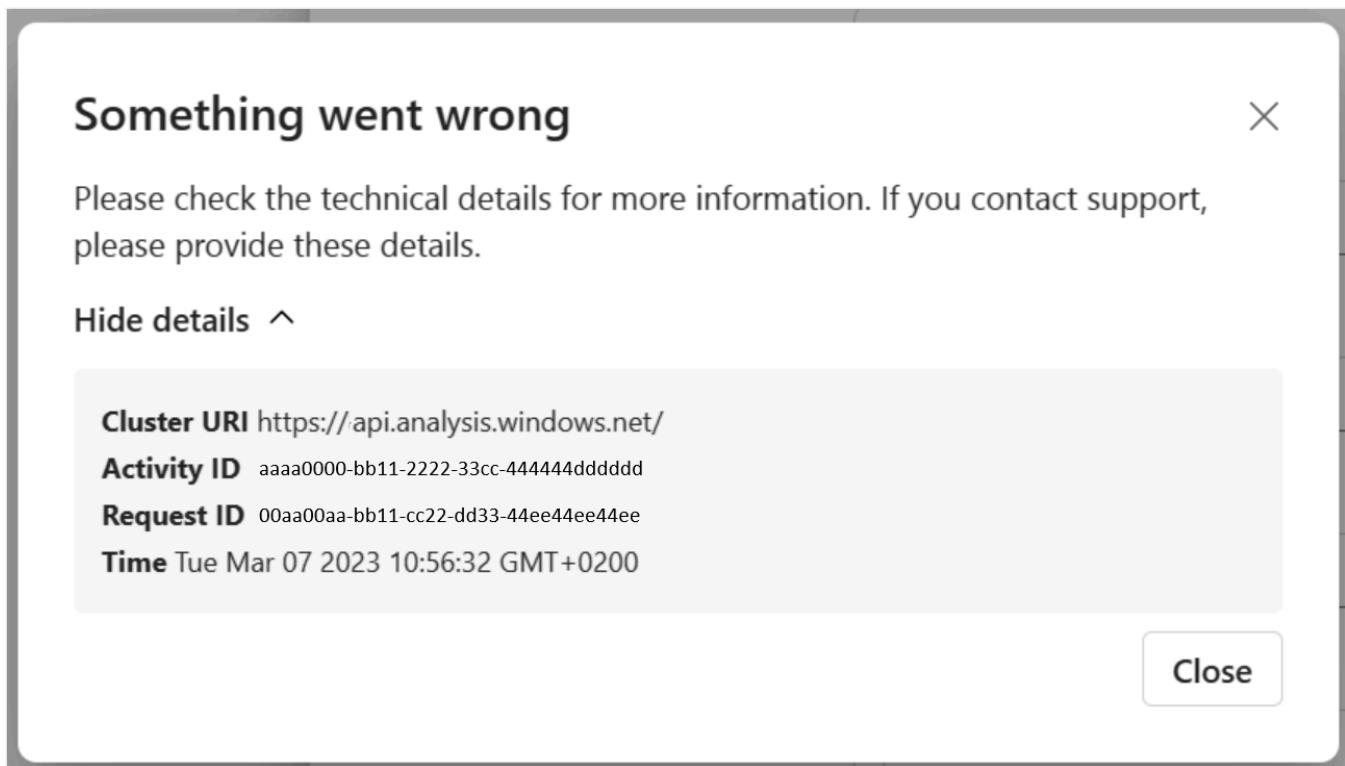
Description of problem: When I try to connect to a Git repo I get a message that it can't connect because the workspace is in a different region.

Cause: If the workspace and repo are located in different regions, the cross-region switch must be enabled.

Solution: [Enable Git actions on workspaces residing in other geographical locations](#).

Connect failure: It says something went wrong when I try to connect

Description of problem: After selecting **Connect** in the Git integration tab, the **Something went wrong** error dialog pops up. In addition, when you select the source control button, the pane indicates that you need to sync with the Git branch.



Cause: If the folder you're trying to connect to has subdirectories but no Fabric items, the connection fails.

Solution: Open the Git repository in Azure DevOps and navigate to the Git folder defined in the connection. If the Git folder contains subdirectories, check that at least one of them represents an item directory. If the directory contains item.config.json and item.metadata.json files, it's an item directory. If the directory doesn't contain these files, it's a subdirectory. If the Git folder doesn't contain any item directories, you can't connect to it. Either remove the subdirectories or connect to a different folder that doesn't contain subdirectories.

The Source control icon doesn't have a number

Description of problem: The number on the Source control icon indicates the number of changes that were made to the workspace since the last commit. If the icon doesn't have a number, there might have been a problem connecting to the branch.

Solution: Disconnect and reconnect.



Connect failure: It says I need a Premium license to connect to git

Description of problem: My workspace was previously connected to a Git repo, but now it says that I need a premium license to connect.

Cause: You can only connect to Git repos if you have a valid Premium license. If your license expired or you change your license to a license that doesn't include Git integration, you won't be able to connect to that repo anymore. This applies to trial licenses as well.

Solution: Disconnect from Git and work without source control, or purchase a Premium license.

Branching out: I don't see the branch I want to connect to

Description of problem: I don't see the workspace I want to connect to in the branching out tab of the **Source control** panel.

Cause: The branching out list only shows workspaces that you have permission to view.

Solution: Check that the workspace you want exists and that you have permission to view it. If not, ask the owner of the workspace to give you permission. See [Branch limitations](#) for more information.

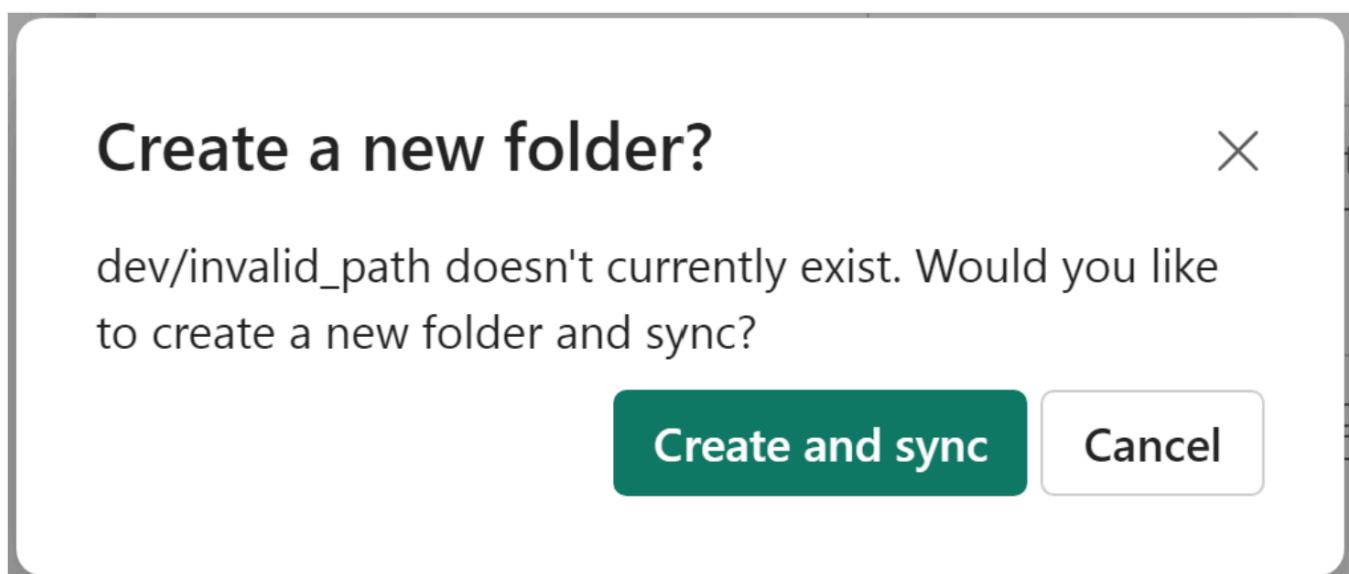
Branching out: My new workspace wasn't synced with my Git repository

Description of problem: When branching out to a new workspace, I'm navigated to the new workspace but Git integration isn't enabled there. **Cause:** The [Git integration switch](#) might be enabled for your source workspace, but not for the whole tenant as the tenant admin can delegate control of the switch to workspace admins. If this is the case, your new workspace won't have Git integration enabled and you'll need to manually enable it from the workspace settings before syncing the workspace with Git. **Solution:** Enable Git integration from the workspace settings of your new workspace.

Connect folder issues

Connect failure: It's asking if I want to create a new folder when I try to connect to a Git branch

Description of problem: After selecting **Connect** in the Git integration tab, a dialog pops up indicating an invalid folder path.



Cause: The folder you're trying to connect doesn't exist, was deleted, or differs in case sensitivity from existing folders in the repository. This message can appear if you're connecting to a new branch, or if the folder was deleted from the branch.

Solution:

- To create a new folder and connect it to the workspace, select **Create and sync**.
- To connect the workspace to a different folder, select **Cancel** and choose another folder in the workspace settings of the Git integration tab.

My Git status says I have uncommitted changes, but I didn't make any changes to my workspace

Description of problem: I want to update my workspace but it says that I have uncommitted changes. I didn't make any changes to my workspace.

Cause: If your workspace has folders and the connected Git folder doesn't yet have subfolders, they are considered to be different. If your workspace has folders but the Git branch doesn't, you see the *uncommitted changes* message. If you try to update the workspace before committing the changes, you get a conflict. Once the Git folder has the same folder structure as the workspace, you won't get this message anymore.

Solution: To resolve the issue, [commit](#) changes to Git. If you can't make changes directly to the connected branch, we recommend using the [checkout branch](#) option. For more information, see [Handling folder changes safely](#).

Commit issues

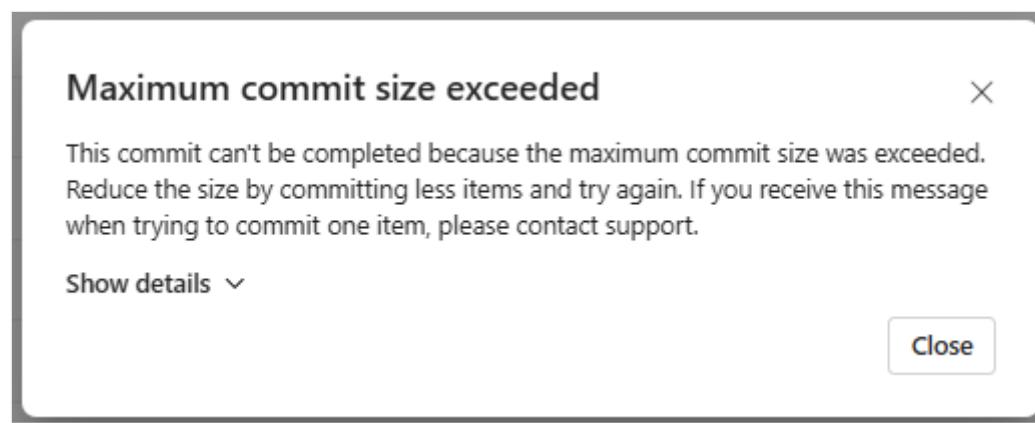
The Commit button is disabled

Description of problem: If there were updates made to the Git branch, commits are disabled until you update your workspace.

Solution: To enable commits, update your workspace.

Maximum commit size exceeded

Description of problem: When trying to commit items to Git, I get an error saying that I exceeded maximum commit size.



Cause: The total size of files to commit is limited to 50 MB.

Solution: If you're trying to commit several items at once, consider committing them in smaller batches. If your commit contains one item with many files, contact [support](#).

Update issues

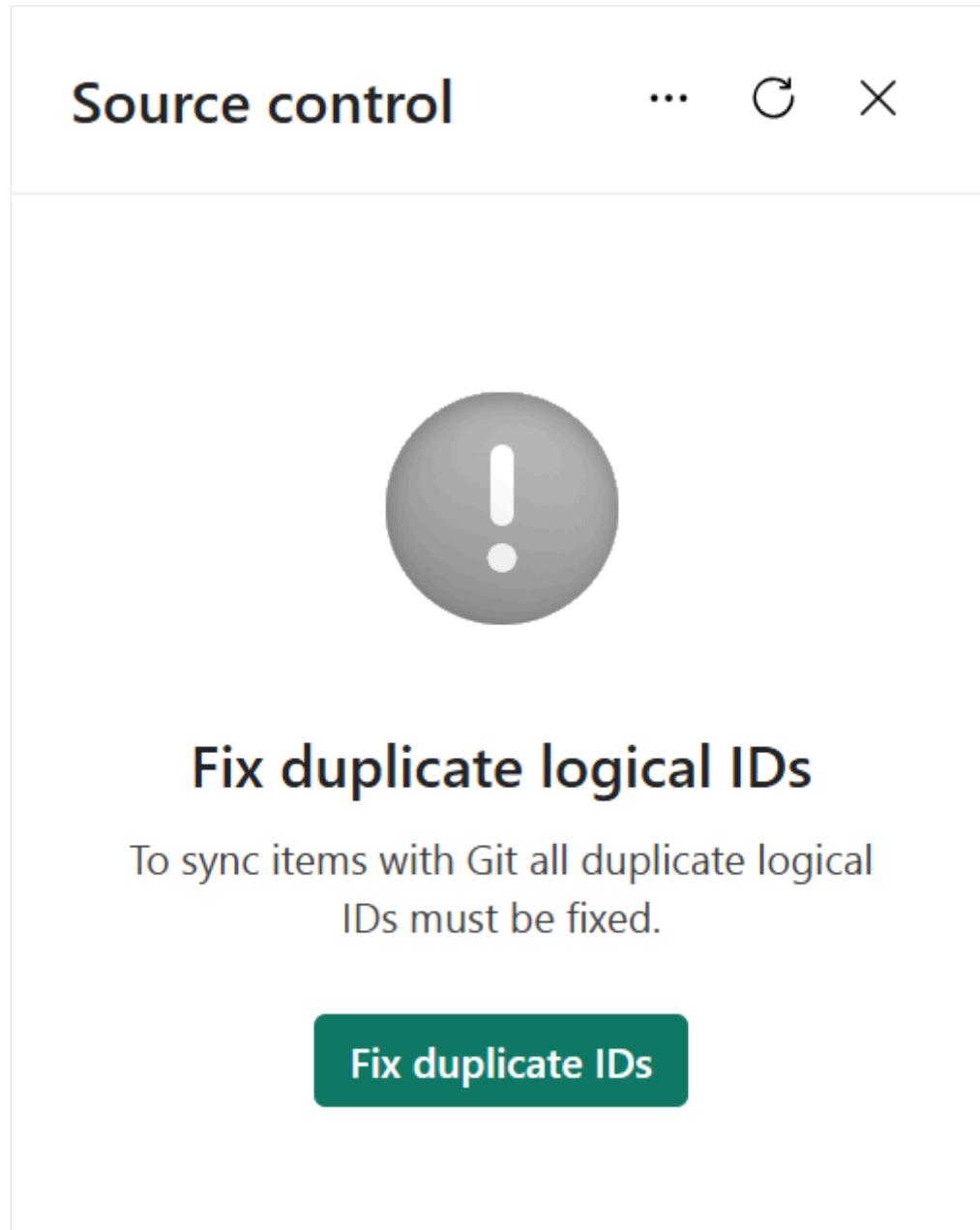
The Commit and Update buttons are both disabled

Description of problem: Changing the same item in the workspace and the Git branch can lead a possible conflict. If changes were made in the workspace and in the Git branch on the same item, updates are disabled until the conflict is resolved.

Solution: [Resolve conflicts](#) and then try again.

Update failure: Fix duplicate logical IDs

Description of problem: When trying to update, a dialog pops up indicating failure because the Git directory contains items with duplicate logical IDs.



Cause: The [logical ID](#) of each item in the workspace must be unique. When you copy an item in the workspace, the logical ID is automatically changed to a unique ID. When you copy an item's directory in Git, the logical ID isn't changed. If you copy an item file in Git, and then try to update to the workspace, the logical ID is duplicated, causing an error.

Solution: To resolve the issue, you need to change the logical ID of the duplicate item or items in Git before updating the workspace. You have two options:

Fix duplicate logical IDs

The following items have duplicate logical IDs:

 Notebook 2

You can fix the issue with a direct commit to the repository if you have permission. If you don't have permission to make a direct commit you need to create a new branch and submit a pull request with the Git provider [Learn more](#).

[Fix with direct commit](#) [Create branch and go to Git](#)

- If you have permission to make direct commits to the branch, select **Fix with direct commit**. This will modify the item's system file to create a unique logical ID in Git. The workspace data isn't modified until you update from Git.
- If you don't have permission to make direct commits to the branch, select **Create branch and go to Git**. This will open a new branch and change the logical ID. You then need to merge the changes in Git before they can be seen in Fabric. Then, when you update from Git, the workspace data is modified.

Update failure: Update doesn't complete because it would break dependency links

Description of problem: After selecting **Update all** or **Undo**, a dialog pops up indicating failure because the action would break a dependency links.

Unable to complete action

X

We can't complete this action because it will break dependency links.

Hide details ^

Cluster URI <https://analysis.windows.net/>

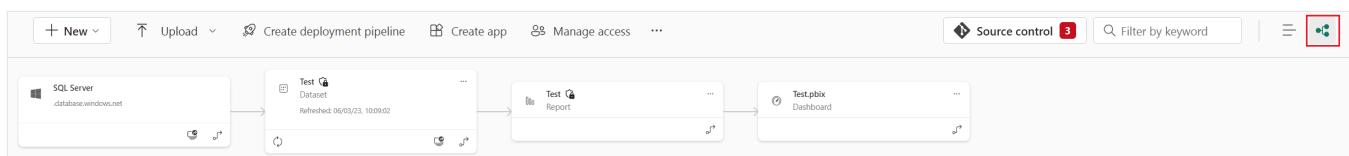
Activity ID aaaa0000-bb11-2222-33cc-444444dddddd

Request ID 00aa00aa-bb11-cc22-dd33-44ee44ee44ee

Time Tue Apr 04 2023 13:09:02 GMT+0300

[Close](#)

Solution: Open the Lineage view to find the item or items that would be deleted from the workspace in the update and are linked to items that won't be deleted from the workspace.



To resolve the issue, delete the problematic item(s):

- If the item isn't [supported by Git](#) (for example, Dashboards), delete it manually from the workspace.
- If the item is [supported by Git](#) (for example, reports), delete it either from Git (if it exists) or from the workspace.

Select **Update All**.

For more information, see [Manually Update from Git](#).

Post update failure: Dependencies aren't pointing to the correct items

Description of problem: After updating from Git, when looking at the lineage view, the dependencies of some items aren't as expected. For example, the proxy model no longer points to the correct model.

Reason: Git Integration doesn't support Direct Query and proxy models at this time.

Solution: To fix the dependencies, do one of the following actions:

- Edit the *bim* file of the ProxyDataset in the Git repository so that it points to the correct dataset, and then, in the workspace, update from Git to receive the change.
- Use the [Update Datasource API](#) to update the connection details of the proxy model in the workspace.

Resolve error issues

Fix duplicate logical IDs

Description of problem: When you try to commit changes to Git, you get an error message that says that there are duplicate logical IDs in the workspace.

Fix duplicate logical IDs

The following items have duplicate logical IDs:

 Notebook 2

You can fix the issue with a direct commit to the repository if you have permission. If you don't have permission to make a direct commit you need to create a new branch and submit a pull request with the Git provider [Learn more](#).

[Fix with direct commit](#) [Create branch and go to Git](#)

Cause: The logical ID is a unique ID for each item in the workspace. When you copy an item in Git, the entire folder is duplicated exactly, including the logical ID. When you try to update the workspace, the system checks for duplicate logical IDs and prevents you from committing changes if it finds any.

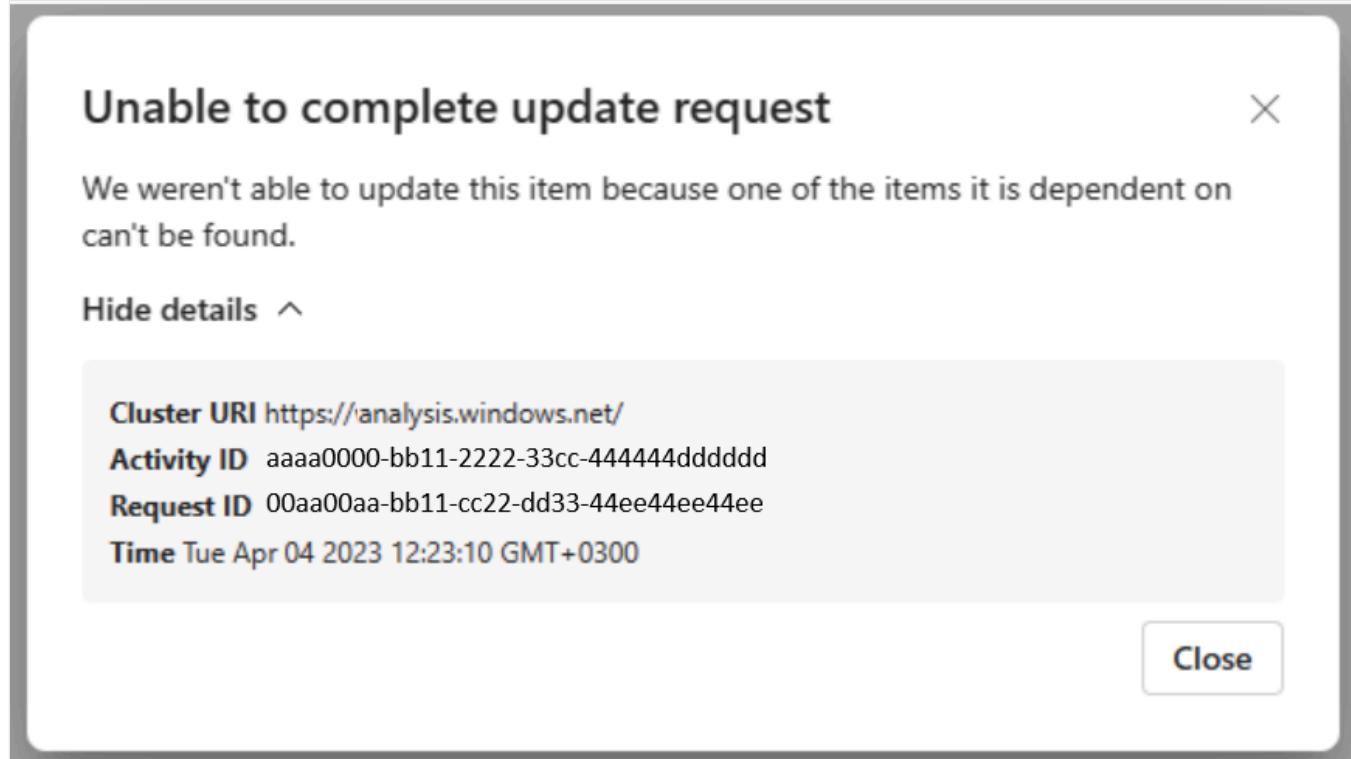
Solution: To fix the issue, you need to change the logical ID of one of the items.

- If you have write permission to the repository, select **Fix with direct commit**. A new branch is automatically created. Change the logical ID of the copied item in the new branch, and then commit the changes.
- If you don't have write permission to the repository, select **Create branch and go to Git***. A new branch is automatically created. Change the logical ID of the copied item in the new branch, and then create a pull request to merge the changes.

Undo issues

Undo failure: After selecting "Undo" a dialog pops up indicating failure because dependency can't be found

Description of problem: The following error appears after an undo action if there's an uncommitted dependency in the **Changes** tab that wasn't selected in the "Undo" action.



Solution: Select the all the dependencies of the selected database and try again.

Dependency error: After selecting "Undo", "Update", or "Switch branch" a dialog pops up indicating failure because the action would break a dependency link

Description of problem: The following error appears after an undo, update, or switch branch action:

Unable to complete action

X

We can't complete this action because it will break dependency links.

Hide details ^

Cluster URI <https://analysis.windows.net/>

Activity ID aaaa0000-bb11-2222-33cc-444444dddddd

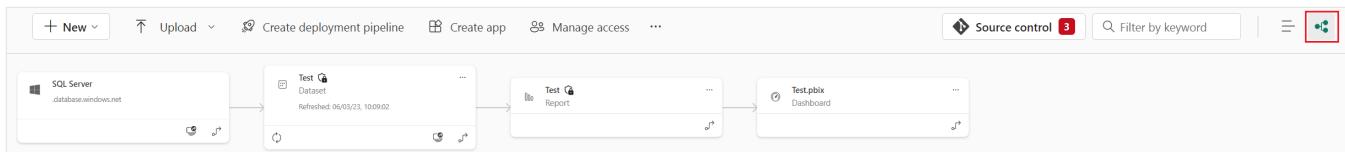
Request ID 00aa00aa-bb11-cc22-dd33-44ee44ee44ee

Time Tue Apr 04 2023 13:09:02 GMT+0300

[Close](#)

Cause: There's an unsupported item in the workspace that depends on an item that's no longer in the workspace causing a dependency problem.

Solution: Open the [Lineage view](#) to find the item or items that were selected to be "undone" and are linked to items that aren't selected.



To resolve the issue, delete the problematic item(s):

- If the item that's not selected is supported by Git (for example, reports), select it to be deleted as well.
- If the item that's not selected isn't supported by Git (for example, Dashboards), [delete it manually](#) from the workspace.

To read more about dependencies, see [Understand dependencies](#).

Deployment pipelines

I can't see the deployment pipelines button

The following conditions must be met in order to see the deployment pipelines button.

- You have a [Fabric license](#).

- You're an admin of a [workspace](#).
- You have [admin permissions](#) for the deployment pipeline the workspace is assigned to.

I can't see the pipeline stage tag in my workspace

Deployment pipelines display a pipeline stage tag in workspaces that are assigned to a pipeline. To see these tags, you need to be a [pipeline admin](#). Tags for the *Development* and *Test* stages are always visible. However, you only see the *Production* tag if you have [access to the pipeline](#).



Lost connections after deployment

Description of problem: In a full pipeline, after you unassign a workspace from a stage and then deploy to it, deployment pipelines reestablishes the connections between items in the source stage you deployed from and the target stage. However, sometimes deployment pipelines can't reestablish the connections between items in the source and target stages. This can happen, for example, when you accidentally delete an item.

Solution: To reestablish these connections, unassign and reassign the same workspace in the target stage.

I can't assign a workspace to a stage

Cause: When you assign a workspace to a deployment pipelines stage, deployment pipelines checks the items (such as reports and dashboards) in the workspace. If there are two items of the same type with the same name in an adjacent stage, deployment pipelines can't determine which one of them should match the one in the assigned workspace, and the **can't assign the workspace** error message appears. For example, if you're trying to assign a workspace to the *test stage*, and one of your reports is called "regional sales", if there's more than one report with the same name in either the *development* or *production* stages, the assignment fails. Assigning your workspace will also fail if the workspace you're assigning has two semantic models titled "regional sales semantic model", and there's a semantic model with the same name in either the *development* or *production* stages.

Solution: To resolve this error, change the name of the item that doesn't match the item in the stage you're trying to assign. You can select the links in the error message to open the items in Fabric.

 **Can't assign the workspace** X

Items from the same category can't have identical names. Update the item names and reassign the workspace. [Learn more](#)

-  ODataFeed-Anonymous (Development)
-  ODataFeed-Anonymous (Development)
-  ODataFeed-Anonymous (Test)
-  ODataFeed-Anonymous (Test)
-  ODataFeed-Anonymous (1) (Development)

Please check the technical details for more information. If you contact support, please provide these details.

[See details](#) ▾

Close

I see the 'different' symbol after I assigned a workspace with semantic models that are similar to the semantic models in adjacent stages

Cause: Most semantic models use the [enhanced semantic model metadata](#) feature, also known as *model v3*. However, older reports might be using the old type of semantic model metadata, sometimes referred to as *model v1*. If you're assigning a workspace that uses the old semantic model metadata model (v1), deployment pipelines can't evaluate whether the semantic model is similar in adjacent stages. In such cases, the *different* UI symbol is displayed, even when the semantic models are identical.

Solution: To resolve this issue, deploy the semantic models that are showing the *different* symbol.

I can't see all my workspaces when I try to assign a workspace to a pipeline

Cause: There can be several reasons why you can't see a workspace in the list of workspaces you can assign to a pipeline.

Solution: To assign a workspace to a pipeline, the following conditions must be met:

- You're an admin of the workspace
- The workspace isn't assigned to any other pipeline
- The workspace resides on a [Fabric capacity](#)

Workspaces that don't meet these conditions, aren't displayed in the list of workspaces you can select from.

My first deployment failed

Cause: Your first deployment might have failed for any of several reasons.

Solution: Some possible reasons for failure with their solutions are listed in the following table.

[] [Expand table](#)

Error	Action
You don't have capacity permissions .	If you work in an organization that has a Fabric capacity, ask a capacity admin to add your workspace to a capacity, or ask for assignment permissions for the capacity. After the workspace is in a capacity, redeploy. If you don't work in an organization with a Fabric capacity, consider purchasing Premium Per User (PPU) .
You don't have workspace permissions.	To deploy, you need to be a workspace member. Ask your workspace admin to grant you the appropriate permissions.
Your Fabric admin disabled the creation of workspaces.	Contact your Fabric admin for support.
You're using selective deployment and aren't selecting all the linked items.	Do one of the following: Unselect the content that is linked to your semantic model or dataflow. Your unselected content (such as semantic models, reports, or dashboards) won't be copied to the next stage.

Error	Action
	Select the semantic model or the dataflow that's linked to the selected items. Your selected items will be copied to the next stage.

I have 'unsupported items' in my workspace when I'm trying to deploy

Cause: Deployment pipelines doesn't support all items.

Solution: For a comprehensive list of supported items that in deployment pipelines, see the following sections:

- [Supported items](#)
- [Item properties that aren't copied](#)

Any item not listed in the supported items list isn't copied to the next stage.

I want to change the data source in the pipeline stages

Cause: You can't change the data source connection in Power BI service.

Solution: If you want to change the data source in the test or production stages, you can use [deployment rules](#) or [APIs](#). Deployment rules will only come into effect after the next deployment.

I fixed a bug in production, but now the 'deploy to previous stage' button is disabled

Cause: You can only deploy backwards to an empty stage. If you have content in the test stage, you can't deploy backwards from production.

Solution: After creating the pipeline, use the development stage to develop your content, and the test stages to review and test it. You can fix bugs in these stages, and then deploy the fixed environment to the production stage.

!¹ Note

Backwards deployment only supports [full deployment](#). It doesn't support [selective deployment](#)

Error message: 'continue the deployment'

Cause: Source stage schema breaking changes, such as replacing a column type from an integer to a string, cause data loss in the target semantic model after deployment.

During deployment, the metadata in the source semantic model is checked against the target metadata. Schema breaking changes cause the deployment to stop. When this happens, you receive the *continue the deployment* message.

Continue the deployment?

Changes to the metadata in the source dataset caused a conflict with the data in the target dataset. Deployment can't continue with the data that's currently in the target dataset.

If you continue this deployment, all the data currently in the target dataset will be deleted. When deployment completes, the target dataset will remain empty until it's refreshed.

operation ID: 3e8401b6-4a56-4169-9cad-e3e3dcacc956

 Family

Do you want to continue this deployment?

See details ▾

Continue Close

Solution: If you continue with the deployment, you lose the data in the target stage. You can use this option if the changes you made to the semantic model were intentional. After the deployment completes, you'll need to refresh the target semantic model.

If the changes weren't intentional, close the message window, upload a fixed .pbix file to the source workspace and redeploy.

After a deployment fails due to schema changes, the target stage displays the *Deployment failed* message, followed by the *Show details* link. The link opens the same *continue the deployment* message that was displayed during the failed deployment.

Error message: 'can't start the deployment'

Cause: When you're using [incremental refresh](#), only certain [changes to the semantic model](#) you're deploying are allowed. If you made semantic model changes that aren't allowed, your deployment fails and you receive this message:

 **Can't start the deployment** ×

Some incrementally refreshed tables in the source dataset, or some of their columns, were renamed. This would result in a data loss to existing data in the target dataset.

Republish either the source or destination dataset to ensure no data loss is caused by this deployment. Then try deploying the content again.

 [IrTestForBug5Days1Year](#)

Please check the technical details for more information. If you contact support, please provide these details.

See details ▾

Close

Solution: If you made changes to your semantic model intentionally, use one of the following workarounds:

- **Using .pbix** - Publish your changes directly to the target semantic model. All partitions and data are lost, so you need to refresh the semantic model.
- **Using XMLA tools** - Make your changes directly on the semantic model in the target stage.

My visual broke after deploying a semantic model or a dataflow

Cause: Semantic model and dataflows are Fabric items that store data and contain both data and metadata. During deployment, only the metadata is copied while the data isn't. As a result, after deployment the semantic model or dataflow might not have any data and a report visual that's relying on this data, will appear broken.

Solution: To solve this problem, refresh the dataflow and then refresh the semantic model in the target stage.

How can I delete a pipeline that doesn't have an owner (an orphaned pipeline)?

Cause: When working with deployment pipelines, you might end up with a pipeline that doesn't have an owner. For example, a pipeline can be left without an owner when a user that owned it leaves the company without transferring ownership. When a pipeline doesn't have an owner, other users can't access it. As a workspace can only be assigned to one pipeline, if it's assigned to a pipeline without an owner, nobody is able to unassign it, and you can't use the workspace in another pipeline.

Solution: When a pipeline is left without an owner, a Fabric administrator can add a new owner to the pipeline, or delete it. To add an owner to the pipeline, use the [Admin - Pipelines UpdateUserAsAdmin API](#).

You can also review our PowerShell script, [AddUserToWorkspacePipeline](#) (available from the [PowerBI-Developer-Samples](#) GitHub repository), which lets you do the following:

- *Manage pipeline access* - Add any user to a workspace in a pipeline.
- *Reclaim workspace ownership* - Add any user to a workspace in a pipeline that doesn't have an owner, allowing you to unblock it.

To use this script, you need to provide a *workspace name* and a *user principal name (UPN)*. The script finds the pipeline that the workspace is assigned to, and add admin permissions to the user you specified.

Mismatch error: Source and target semantic model format version mismatch error

Description of problem: The *Can't start deployment* error that states that *the source and target semantic models have different data modeling formats*, occurs when the semantic models in the target stage have a higher model version than the semantic models in the source stage. In such cases, deployment pipelines aren't able to deploy from the source stage to the target stage. To avoid this error, use a semantic model that has the same (or higher) model version in the source stage.

Solution: Upgrade the semantic model in the source stage using an [XMLA read-write endpoint](#) or Power BI Desktop. After upgrading the semantic model, republish it to the source stage.

Mismatch error: Data source connectivity mode mismatch error

Description of problem: During deployment, if deployment pipelines discovers that the connectivity mode of a data source in the target stage isn't the same as the data source in the source stage, it attempts to convert the connectivity mode of the data source in the target stage. If you're using a data source with the [live connection](#) or [real time](#) connectivity modes, deployment pipelines can't convert the target's data source connectivity mode.

Solution: Either use an [XMLA read-write endpoint](#) or Power BI Desktop to change the connection mode of the data source in the source stage, or delete the data source in the target stage so that the deployment overwrites it.

My semantic model deployment failed

Cause: There could be a few possible reasons for your semantic model deployment to fail. The following are possible reasons for failure:

- A large semantic model isn't configured with the [large semantic model format](#).
- The semantic model contains a circular or self dependency (for example, item A references item B and item B references item A). In this case you'll see the following error message: *One or more items failed to deploy because it will result in a two way dependency between items.*

Solution:

- If your semantic model is larger than 4 GB and isn't using the large semantic model format, it might fail to be deployed. Try setting your semantic model to use the large semantic model format, and redeploy.
- If your semantic model contains a circular or self dependency, remove the dependency and redeploy.

I have a semantic model with DirectQuery or Composite connectivity mode that uses variation or auto date/time tables

Cause: Semantic models that use DirectQuery or Composite connectivity mode and have variation or [auto date/time](#) tables aren't supported in deployment pipelines.

Solution: If your deployment fails and you think it's because you have a semantic model with a variation table, you can look for the [variations](#) property in your table's columns. You can use one of the following methods to edit your semantic model so that it works in deployment pipelines.

- Use [import](#) mode instead of *DirectQuery* or *Composite* mode in your semantic model.

- Remove the [auto date/time](#) tables from your semantic model. If necessary, delete any remaining variations from all the columns in your tables. Deleting a variation might invalidate user authored measures, calculated columns, and calculated tables. Use this method only if you understand how your semantic model works as it could result in data corruption in your visuals.

Paginated reports

I can't deploy a paginated report

Solution: To deploy a paginated report, you need to be a workspace member in the workspace you're deploying from (the source stage workspace). If you're not a workspace member in the source stage, you can't deploy the paginated report.

Data source mismatch: Target stage paginated report displays data from a Fabric semantic model in the source stage

Description of problem: Currently, semantic models are treated as an external Analysis Services data source, and connections to semantic models aren't switched automatically after deployment.

When you deploy a paginated report that's connected to a Fabric semantic model, it continues to point to the semantic model it was originally connected to. Use [deployment rules](#) to point your paginated report to any semantic model you want, including, for example, the target stage semantic model.

Solution: If you're using a paginated report with a Fabric semantic model, see [How do I create a deployment rule for a paginated report with a Fabric semantic model?](#)

Deployment failure: Large number of paginated reports fails

Description of problem: A deployment of a large number of paginated reports with rules might fail due to an overload on the capacity.

Solution: Either purchase a higher [SKU](#), or use selective deployment.

Dataflows

Lineage view: I deleted a data source that belonged to a dataflow, but I can still see it in the lineage view

Cause: In dataflows, old data sources aren't removed from the dataflow data source page. To support the dataflows lineage view, connected items aren't deleted.

Solution: This behavior doesn't affect deployment pipelines. You can still refresh, edit, and deploy dataflows in a pipeline.

I see two data sources connected to my dataflow after using dataflow rules

Description of problem: After changing a dataflow's data source using a rule, the dataflow's lineage view displays a connection between the dataflow's source data source, and the data source configured in the rule.

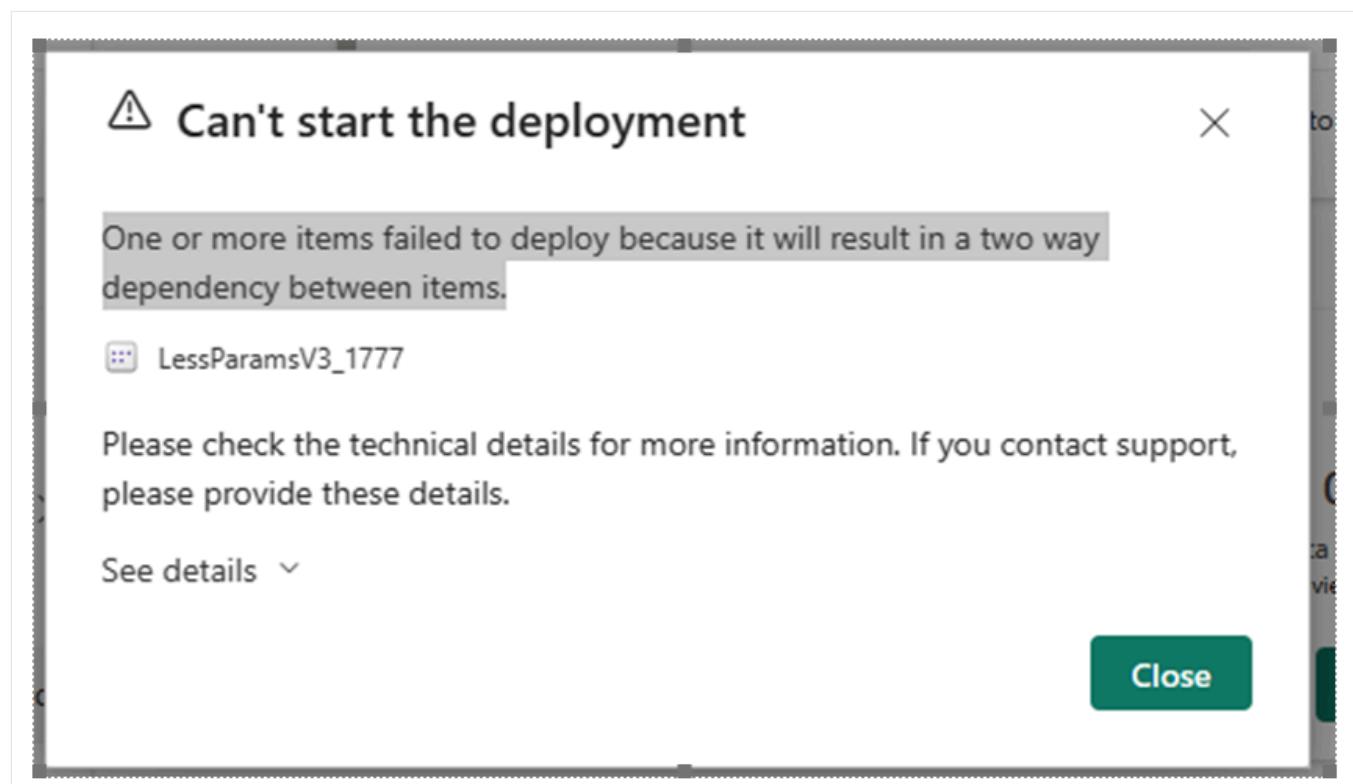
Solution: This behavior doesn't affect deployment pipelines.

Datamarts

Deployment problem: I can't deploy a datamart in the pipeline

Solution: To deploy a datamart, you must be the owner of the datamart.

Deployment problem: My datamart deployment failed because of a circular dependency



Solution: There's either an item that references itself, or more than one item involved in a circular chain of references (for example, item A references item B and item B references item A). To deploy the datamart, remove the circular dependency and redeploy.

Permissions

Who can deploy content between stages?

Content can be deployed to an empty stage or to a stage that contains content. The content must reside on a [Fabric capacity](#).

- **Deploying to an empty stage** - Any [licensed Fabric](#) user who's a member or admin in the source workspace.
- **Deploying to a stage with content** - Any [licensed Fabric](#) user who's a member or admin of both workspaces in the source and target deployment stages.
- **Overwriting a semantic model** - Deployment overwrites each semantic model that is included in the target stage, even if the semantic model wasn't changed. Any user who's a member or admin of both workspaces, but the tenant admin can restrict this to target semantic model owners only.

I can't see a workspace in the pipeline

Cause: Pipeline and workspace permissions are managed separately. You might have pipeline permissions, but not workspace permissions.

Solution: For more information, review the [permissions](#) section.

Error message: 'workspace member permissions needed'

Solution: To assign a workspace, you need at least [workspace member](#) permissions for the workspaces in its adjacent stages. Workspace member (or higher) permissions in the adjacent stages are required to enable deployment pipelines to establish connections between items in neighboring pipeline stages.

 **Test** [Learn more](#)



Test and verify your content in a preproduction workspace. When it's ready to distribute, deploy the content to the production stage.



Workspace member permissions needed

To assign a workspace to this stage, you need to be a member in Development, so content can be matched. [Learn more](#)

Rules

Deployment failure due to broken rules

Solution: If you have problems configuring deployment rules, visit [deployment rules](#), and make sure you follow the [deployment rules limitations](#).

If your deployment was previously successful, and is suddenly failing with broken rules, it could be due to a semantic model being republished. The following changes to the source semantic model result in a failed deployment:

Parameter rules

- A removed parameter
- A changed parameter name

Data source rules

Your deployment rules are missing values. This might have happened if your semantic model changed.

Invalid rules were found

X

One or more datasets you're deploying have rules that are missing values or aren't valid.

We recommend reviewing and fixing any invalid rules before continuing the deployment.

[Configure rules](#)

[Cancel](#)

When a previously successful deployment fails due to broken links, a warning is displayed. You can select **Configure rules** to navigate to the deployment rules pane, where the failed semantic model is marked. When you select the semantic model, the broken rules are marked.

To deploy successfully, fix or remove the broken rules, and redeploy.

Deployment problem: I configured rules, but it didn't deploy

Cause: Deployment rules aren't applied immediately after they're configured.

Solution: To apply deployment rules, you have to deploy the semantic models from the source stage to the target stage which includes the created deployment rules. After configuring deployment rules, and before you deploy, the *different* indicator is shown next to the semantic model with the configured rules. This indicates that you need to deploy that semantic model from the source stage to the target stage. Once you deploy, if no other changes were made, the *different* indicator disappears signifying that the rules were applied successfully.

Deployment rules are greyed out

Solution: To create a [deployment rule](#), you must be the owner of the item you're creating a deployment rule for. If you're not the owner of the item, deployment rules are greyed out.

Create rules defining the data sources for this report. These rules will be applied when deploying to this stage. [Learn more](#)

ⓘ Only the report owner can change these settings. Go to the [report settings page](#) to take over this report.

▶ Data source rules

▶ Parameters rules

If one of the rule options is greyed out, it could be because of the following reasons:

- **Data source rules** - There are no data sources that a rule can be configured on.
- **Parameters rules** - There are no parameters a rule can be configured for.

My data source rule for a semantic model failed

Solution: Saving data source rules might fail due to one of these reasons:

- Your semantic model contains a function connected to a data source. In such cases, data source rules aren't supported.
- Your data source is using parameters. You can't create a data source rule for a semantic model that uses parameters. Create a parameter rule instead.

I can't connect to a semantic model when creating a new semantic model rule

Cause: When constructing a semantic model using Power BI Desktop, the connection string can be configured. Later, the semantic model can be published and used by deployment pipelines in Power BI service. When creating the connection in Power BI Desktop, you can specify additional parameters. When specifying the parameters, the semantic model source must be the first parameter listed. If you list any other parameters before the semantic model source, you run into errors in Power BI service. In such cases, when configuring a new semantic model

rule, if you point to a semantic model that wasn't configured properly in Power BI Desktop, deployment pipelines can't create the rule.

Solution: Format the semantic model connection in Power BI Desktop so that the semantic model source appears in the first row. Then, republish the semantic model.

Troubleshooting errors

Use this section to troubleshoot pipeline [rules](#) you created. If you don't see a rule error message name, review the [deployment rule limitations](#) and the [supported data sources for dataflow and semantic model rules](#), and try to reconfigure the rule.

 Expand table

Error message	Solution
Data source rule can't contain a parameter	Your rule can't be applied because the server name or database name referenced in the rule is controlled by a parameter. To change the server or database name, use a parameter rule or remove the controlling parameter from configured item.
Data source execution failure	A rule can't be applied due to a problem retrieving data from the data source. Remove the rule and make sure the semantic model has valid queries. Then try creating the rule again.
Rule property no longer exists	Some of the rule properties configured in the rule no longer exist. Refresh the page and configure the rule again.
Illegal value	A value used in the configured rule isn't valid. Validate the rule's values and try configuring the rule again.
Multiple data sources aren't supported	A semantic model rule can't be applied due to its data source configuration. Either remove the rule, or rewrite the semantic model queries using standard Power BI Desktop tools.
Target semantic model can only be changed by its owner	Your rule will overwrite some semantic models in the destination workspace. You must be the owner of any semantic model that will be overwritten.

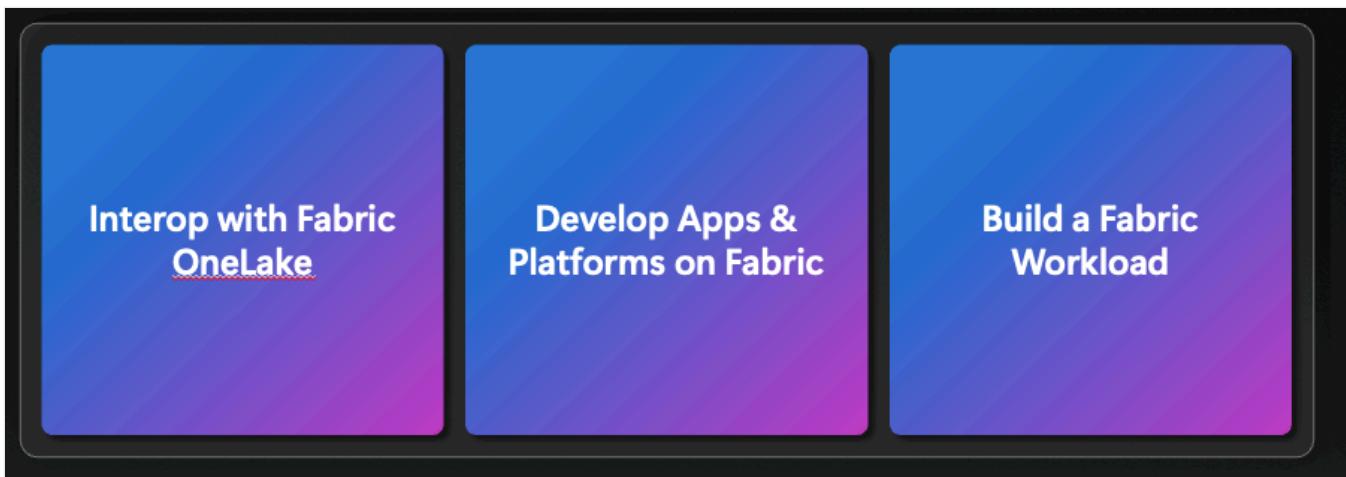
Related content

- [Get started with deployment pipelines](#)
- [Assign a workspace to a pipeline stage](#)
- [Deployment history](#)

Last updated on 12/15/2025

Microsoft Fabric Integration Pathways for ISVs

Microsoft Fabric [↗](#) offers three distinct pathways for Independent Software Vendors (ISVs) to seamlessly integrate with Fabric. For an ISV starting on this journey, we want to walk through various resources we have available under each of these pathways.



Interop with Fabric OneLake

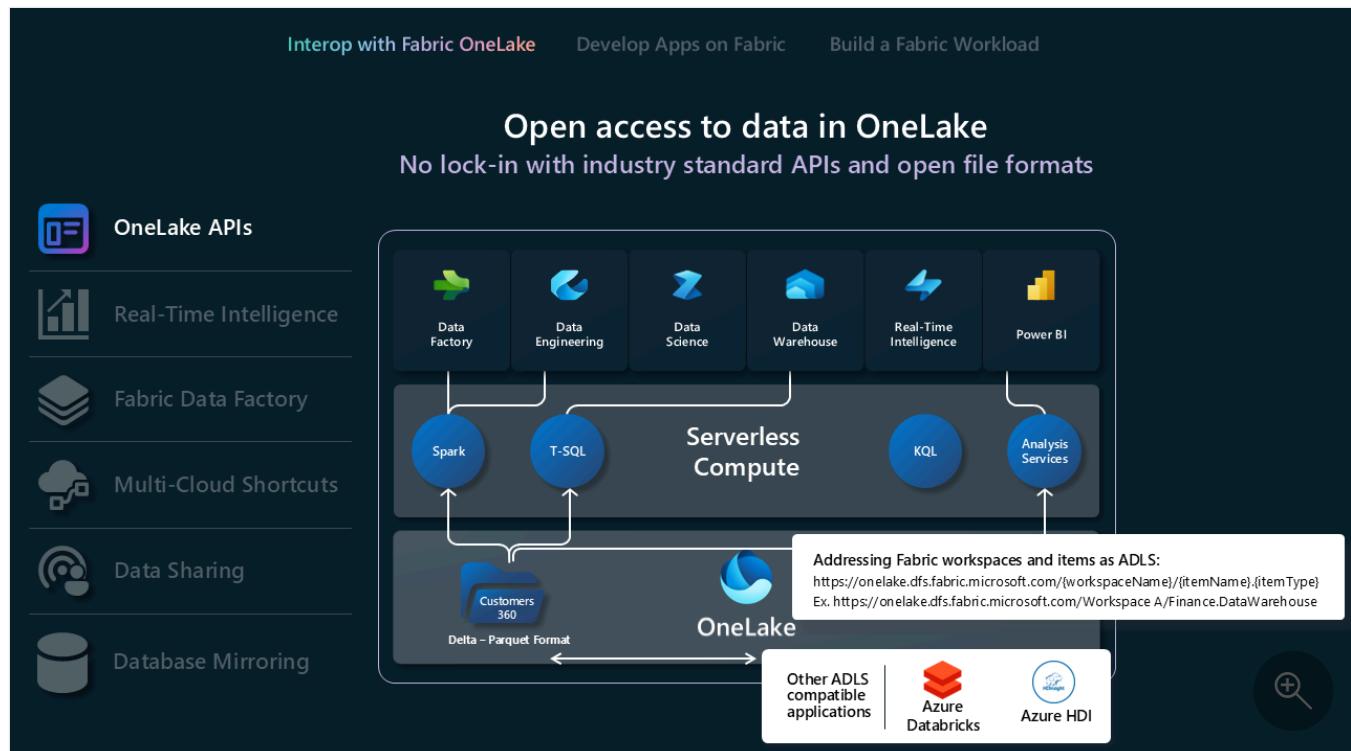
The primary focus with Interop model is on enabling ISVs to integrate their solutions with the [OneLake Foundation](#). To Interop with Microsoft Fabric, we provide integration using a multitude of connectors in Data Factory and in Real-Time Intelligence. We also provide REST APIs for OneLake, shortcuts in OneLake, data sharing across Fabric tenants, and database mirroring.

A dark rectangular card with a title and six items below it. The title is "Top approaches of integrating with OneLake" in white text at the top. Below the title are six dark blue rectangular boxes, each containing an icon and a label. From left to right, the icons are: a square with a minus sign (OneLake APIs), a bar chart with an upward arrow (Real-Time Intelligence), three stacked cubes (Fabric Data Factory), a cloud with a gear (Multi-Cloud Shortcuts), a circular icon with a signal (Data Sharing), and a cylinder (Database Mirroring).

The following sections describe some of the ways you can get started with this model.

OneLake APIs

- OneLake supports existing Azure Data Lake Storage (ADLS) Gen2 APIs and SDKs for direct interaction, allowing developers to read, write, and manage their data in OneLake. Learn more about [ADLS Gen2 REST APIs](#) and [how to connect to OneLake](#).
- Since not all functionality in ADLS Gen2 maps directly to OneLake, OneLake also enforces a set folder structure to support Fabric workspaces and items. For a full list of different behaviors between OneLake and ADLS Gen2 when calling these APIs, see [OneLake API parity](#).
- If you're using Databricks and want to connect to Microsoft Fabric, Databricks works with ADLS Gen2 APIs. [Integrate OneLake with Azure Databricks](#).
- To take full advantage of what the Delta Lake storage format can do for you, review and understand the format, table optimization, and V-Order. [Delta Lake table optimization and V-Order](#).
- Once the data is in OneLake, explore locally using [OneLake File Explorer](#). OneLake file explorer seamlessly integrates OneLake with Windows File Explorer. This application automatically syncs all OneLake items that you have access to in Windows File Explorer. You can also use any other tool compatible with ADLS Gen2 like [Azure Storage Explorer](#).

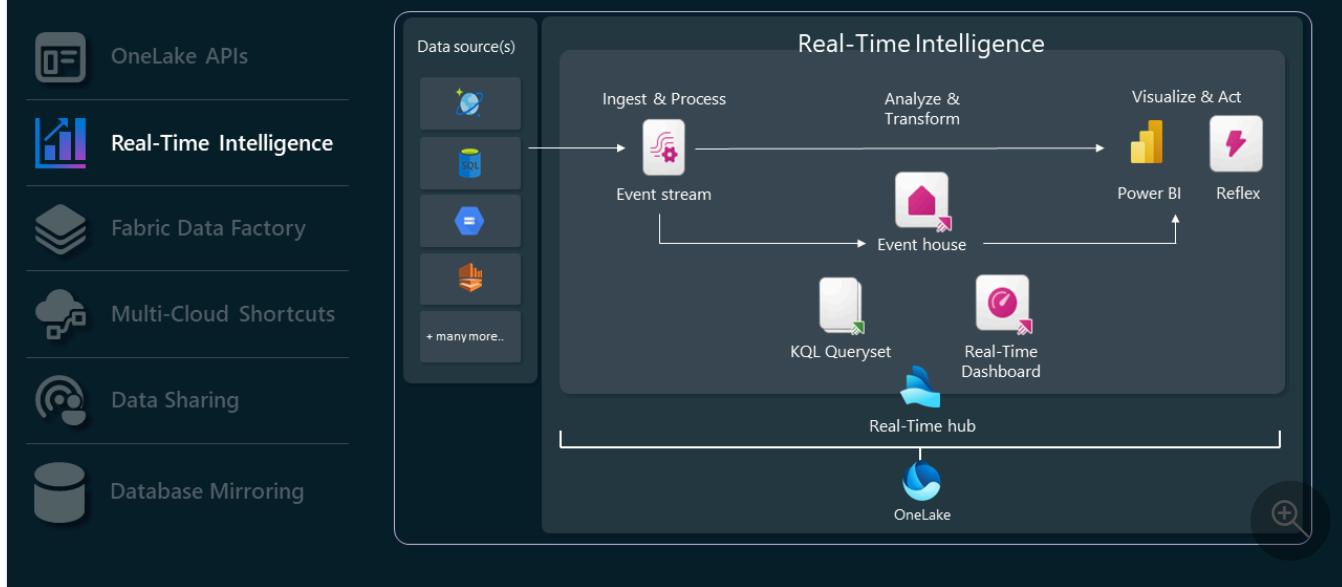


Real-Time Intelligence APIs

Fabric Real-Time Intelligence is a comprehensive solution designed to support the entire lifecycle of real-time data—from ingestion and stream processing to analytics, visualization, and action. Built to handle high-throughput streaming data, it offers robust capabilities for data ingestion, transformation, querying, and storage, enabling organizations to make timely, data-driven decisions.

- **Eventstreams** enable you to bring real-time events from various sources and route them to various destinations, such as Lakehouses, KQL databases in Eventhouse, and Fabric Activator. Learn more about [Eventstreams](#) and [Eventstreams API](#).
- You can ingest streaming data into Eventstreams via multiple protocols incl. Kafka, Event Hubs, AMQP, and a growing list of connectors listed [here](#).
- After processing the ingested events using either the no-code experience or using SQL operator (Preview), the result can be routed to several Fabric destinations or to custom endpoints. Learn more about Eventstreams destinations [here](#).
- **Eventhouses** are designed for streaming data, compatible with Real-Time hub, and ideal for time-based events. Data is automatically indexed and partitioned based on ingestion time, giving you incredibly fast and complex analytic querying capabilities on high-granularity data that can be accessed in OneLake for use across Fabric's suite of experiences. Eventhouses support existing Eventhouse APIs and SDKs for direct interaction, allowing developers to read, write, and manage their data in Eventhouses. Learn more about [REST API](#).
- If you're using Databricks or Jupyter Notebooks, you can utilize the Kusto Python Client Library to work with KQL databases in Fabric. Learn more about [Kusto Python SDK](#).
- You can utilize the existing [Microsoft Logic Apps](#), [Azure Data Factory](#), or [Microsoft Power Automate](#) connectors to interact with your Eventhouses or KQL Databases.
- **Database shortcuts in Real-Time Intelligence** are embedded references within an eventhouse to a source database. The source database can either be a KQL Database in Real-Time Intelligence or an Azure Data Explorer database. Shortcuts can be used for in-place sharing of data within the same tenant or across tenants. Learn more about managing [database shortcuts using the API](#).

Extract insights and visualize data in motion



Data Factory in Fabric

- Pipelines boast an [extensive set of connectors](#), enabling ISVs to effortlessly connect to a myriad of data stores. Whether you're interfacing traditional databases or modern cloud-based solutions, our connectors ensure a smooth integration process. [Connector overview](#).
- With our supported Dataflow Gen2 connectors, ISVs can harness the power of Fabric Data Factory to manage complex data workflows. This feature is especially beneficial for ISVs looking to streamline data processing and transformation tasks. [Dataflow Gen2 connectors in Microsoft Fabric](#).
- For a full list of capabilities supported by Data Factory in Fabric, check out this [Data Factory in Fabric Blog ↗](#).

The screenshot displays the Microsoft Power Query interface within a browser window titled "Data Factory". The interface is used for managing data flows between various sources and destinations. On the left, there's a sidebar with several icons and labels: "OneLake APIs", "Real-Time Intelligence", "Fabric Data Factory", "Multi-Cloud Shortcuts", "Data Sharing", and "Database Mirroring". The main workspace shows a complex data flow with multiple tables and joins. At the bottom, a preview pane displays the resulting data from the query.

Multicloud shortcuts

Shortcuts in Microsoft OneLake allow you to unify your data across domains, clouds, and accounts by creating a single virtual data lake for your entire enterprise. All Fabric experiences and analytical engines can directly point to your existing data sources such as OneLake in different tenant, [Azure Data Lake Storage \(ADLS\) Gen2](#), [Amazon S3 storage accounts](#), [Google Cloud Storage\(GCS\)](#), [S3 Compatible data sources](#), and [Dataverse](#) through a unified namespace. OneLake presents ISVs with a transformative data access solution, seamlessly bridging integration across diverse domains and cloud platforms.

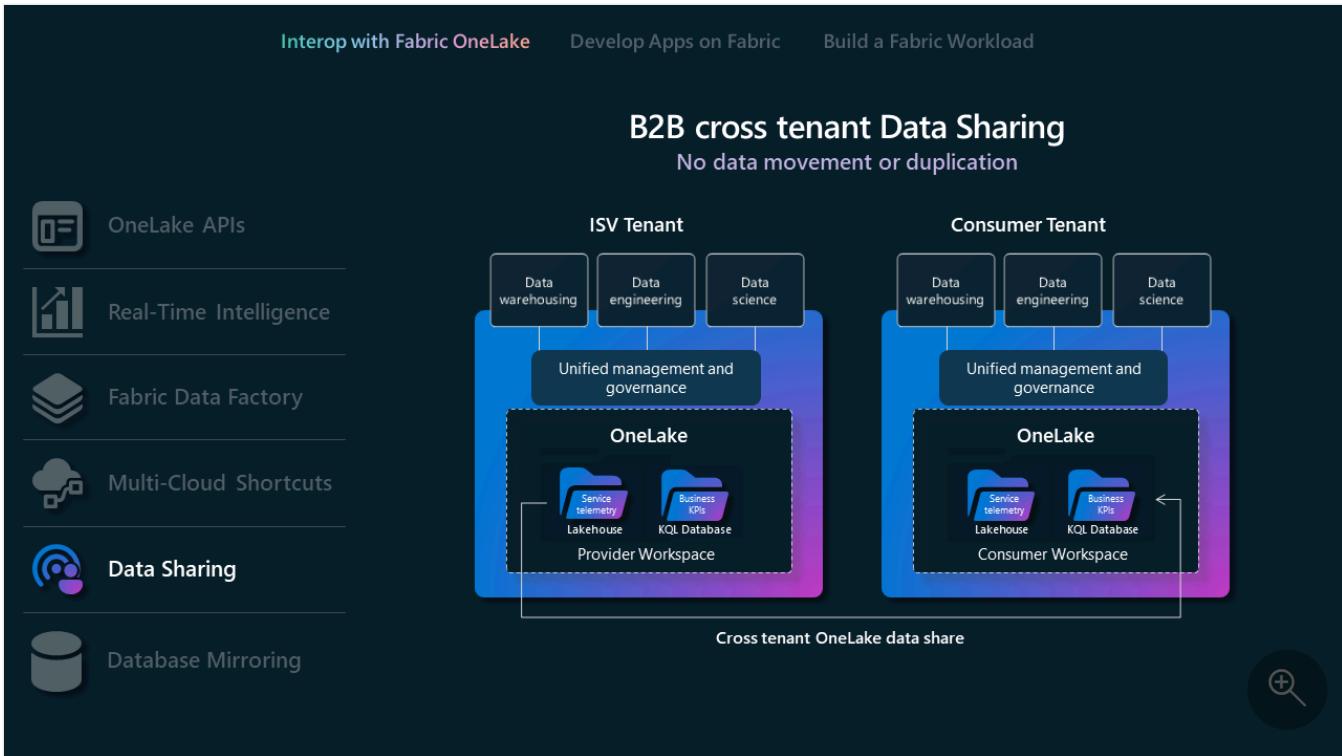
- [Learn more about OneLake shortcuts](#)
- [Learn more about OneLake one logical copy](#)
- [Learn more about KQL database shortcuts](#)



Data sharing

Data Sharing allows Fabric users to share data across different Fabric tenants without duplicating it. This feature enhances collaboration by enabling data to be shared "in-place" from OneLake storage locations. The data is shared as read-only, accessible through various Fabric computation engines, including SQL, Spark, KQL, and semantic models. To use this feature, Fabric admins must enable it in both the sharing and receiving tenants. The process includes selecting data within the OneLake data hub or workspace, configuring sharing settings, and sending an invitation to the intended recipient.

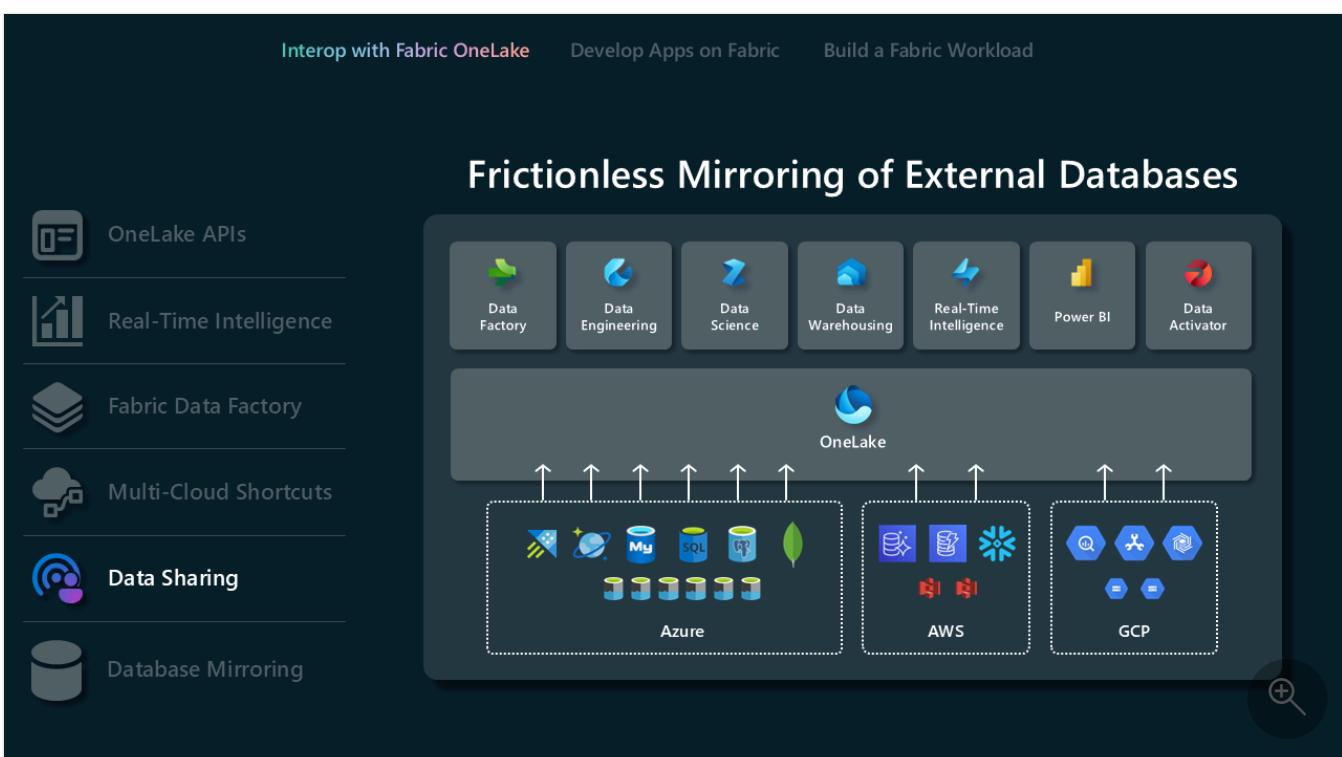
- [Learn more about Data Sharing](#)



Database mirroring

Mirroring in Fabric provides an easy experience to avoid complex ETL (Extract Transform Load) and integrate your existing data into OneLake with the rest of your data in Microsoft Fabric. You can continuously replicate your existing data directly into Fabric's OneLake. In Fabric, you can unlock powerful Business Intelligence, Artificial Intelligence, Data Engineering, Data Science, and data sharing scenarios.

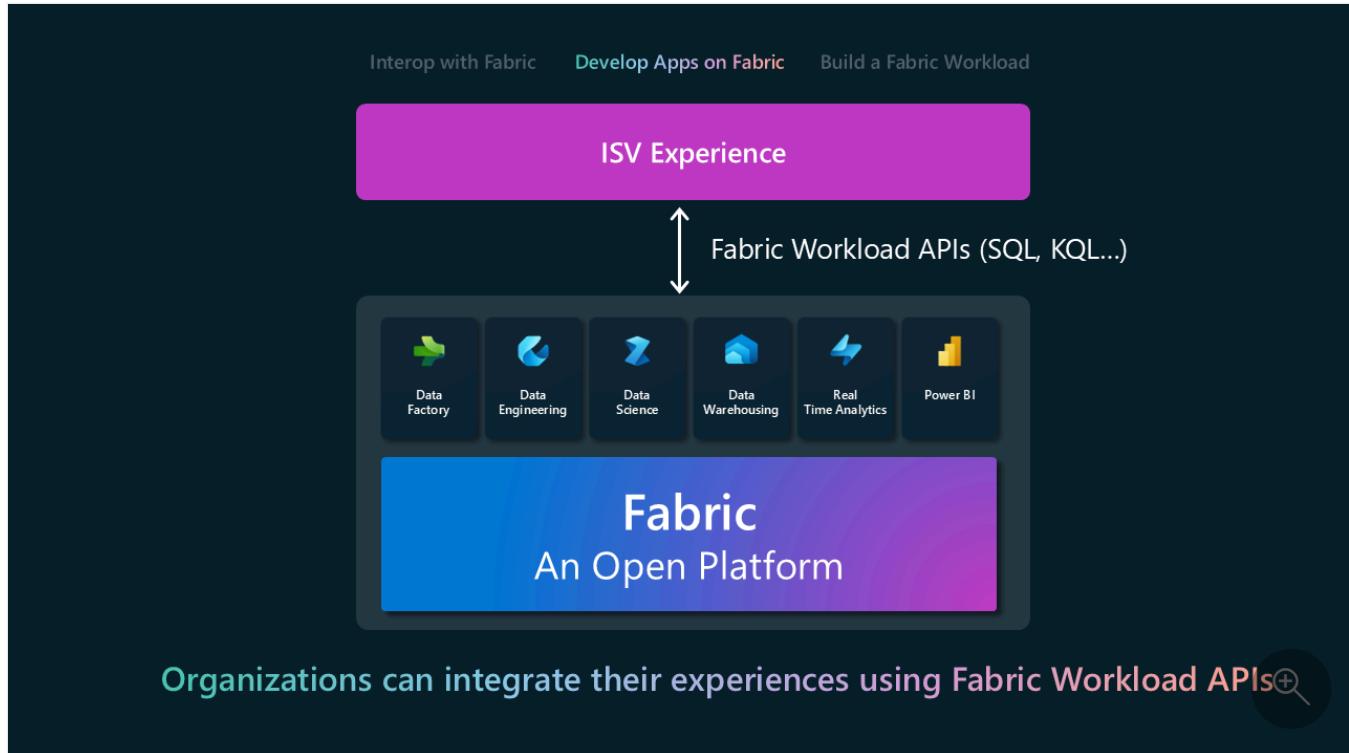
- Learn more about [mirroring and the supported databases](#).



Open mirroring enables **any application** to write change data directly into a mirrored database in Fabric. Open mirroring is designed to be extensible, customizable, and open. It's a powerful feature that extends mirroring in Fabric based on open Delta Lake table format. Once the data lands in OneLake in Fabric, open mirroring simplifies the handling of complex data changes, ensuring that all mirrored data is continuously up-to-date and ready for analysis.

- Learn more about [open mirroring](#) and when to use it.

Develop on Fabric



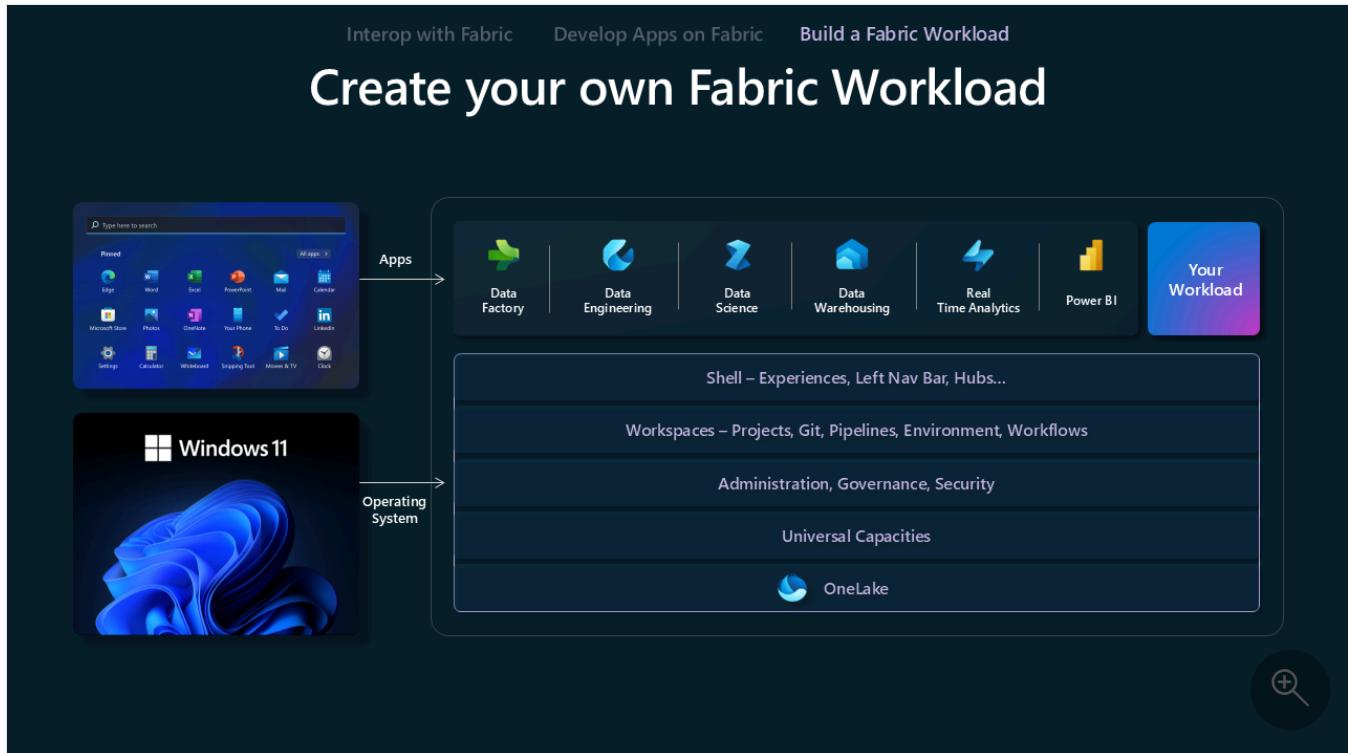
With the **Develop on Fabric** model ISVs can build their products and services on top of Fabric or seamlessly embed Fabric's functionalities within their existing applications. It's a transition from basic integration to actively applying the capabilities Fabric offers. The main integration surface area is via REST APIs for various Fabric experiences. The following table shows a subset of REST APIs grouped by the Fabric experience. For a complete list, see the [Fabric REST API documentation](#).

[] Expand table

Fabric Experience	API
Data Warehouse	- Warehouse - Mirrored Warehouse
Data Engineering	- Lakehouse - Spark - Spark Job Definition

Fabric Experience	API
	<ul style="list-style-type: none"> - Tables - Jobs
Data Factory	- DataPipeline
Real-Time Intelligence	<ul style="list-style-type: none"> - Eventhouse - KQL Database - KQL Queryset - Eventstream
Data Science	<ul style="list-style-type: none"> - Notebook - ML Experiment - ML Model
OneLake	<ul style="list-style-type: none"> - Shortcut - ADLS Gen2 APIs
Power BI	<ul style="list-style-type: none"> - Report - Dashboard - Semantic Model

Build a Fabric workload



Build a Fabric workload model is designed to empower ISVs to create custom experiences on the Fabric platform. It provides ISVs with the necessary tools and capabilities to align their offerings with the Fabric ecosystem, optimizing the combination of their unique value propositions with Fabric's extensive capabilities.

The [Microsoft Fabric Workload Development Kit](#) offers a comprehensive toolkit for developers to integrate applications into the Microsoft Fabric hub. This integration allows for the addition of new capabilities directly within the Fabric workspace, enhancing the analytics journey for users. It provides developers and ISVs with a new avenue to reach customers, delivering both familiar and new experiences, and leveraging existing data applications. Fabric admins have the ability to manage who can add workloads in an organization.

Workload Hub

The [Workload Hub](#) in Microsoft Fabric serves as a centralized interface where users can explore, manage, and access all available workloads. Each workload in Fabric is associated with a specific item type that can be created within Fabric workspaces. By navigating through the Workload Hub, users can easily discover and interact with various workloads, enhancing their analytical and operational capabilities.

The screenshot shows the Microsoft Fabric Workload Hub. On the left is a sidebar with icons for Home, Copilot, Create, Browse, Oracle catalog, Apps, Metrics, Monitor, Learn, Real-Time, Workspaces (which is selected and highlighted with a red box), and My workspace. The main area is titled "Workloads" and contains a section for "Add more workloads". Below this are ten workload cards, each with an "Add" button:

- 2TEST (preview)** by ZBIT: Comprehensive Quality Assurance: Automated Testing and Data Quality checks.
- Informatica Cloud Data Quality (preview)** by Informatica: Informatica's Intelligent Data Management Cloud (IDMC) is the leading platform for cloud data management with Microsoft Fabric, offering data...
- Lumel EPM (preview)** by Lumel: Enterprise Performance Management(EPM) Suite for Planning | PowerTable(Data Apps) | Modern BI
- Neo4j AuraDB with graph analytics (preview)** by Neo4j: Uncover deeper insights from data in OneLake using the Neo4j Workload for Microsoft Fabric, which provides graph analytics for improved decision...
- Osmos** by Osmos: Accelerate your Fabric deployment with Omos AI data agents. Seamlessly ingest, transform, and structure data using agentic AI.
- Power Designer** by PowerBitips LLC: Power Designer enables company wide styling and creation of report templates in record time. Level up your report style by refining your report designs...
- Process Intelligence (preview)** by Celonis: Establish a zero-copy integration with Celonis to enhance your data with process intelligence and expose Celonis' unique class of data and context in...
- Profisee MDM** by Profisee: Match, merge, and standardize data in Microsoft Fabric to create trusted, consumption-ready data products for analytics and AI.
- Quantexa Unify (preview)** by Quantexa: Resolve one or more Microsoft OneLake Data Sources to produce a complete 360° view with unprecedented accuracy.
- SAS Decision Builder (preview)** by SAS Institute Inc.: Automate, optimize, and scale decision-making processes. Manage complex business rules; integrate machine learning models, or use Python code to...
- SQL2Fabric-Mirroring (preview)** by Strim, Inc.: Strim's SQL2Fabric-Mirroring is a fully managed, zero-copy replication solution that seamlessly mirrors on-premises SQL Server data to Microsoft Fabric...
- Statsig (preview)** by Statsig: Visualize and Analyze your data with the Statsig Data Platform.
- Teradata AI Unlimited (preview)** by Teradata: Power faster innovation with Teradata. Teradata AI Unlimited serverless engine via Microsoft Fabric accelerates innovation.

A magnifying glass icon is located in the bottom right corner of the workload grid.

Fabric administrators have the rights to [manage workload](#) availability, making them accessible across the entire tenant or within specific capacities. This extensibility ensures that Fabric remains a flexible and scalable platform, allowing organizations to tailor their workload environment to meet evolving data and business requirements. By integrating seamlessly with Fabric's security and governance framework, the Workload Hub simplifies workload deployment and management. Every workload comes with a trial experience for users to quickly get started. Following are the available workloads:

- **2TEST**: A comprehensive quality assurance workload that automates testing and data quality checks.

2TEST - Automated Data Testing and Data Quality checks (preview)

Comprehensive Quality Assurance: Automated Data Testing and Data Quality checks.

[Add Workload](#)

About

Item types



Compatible with



Publisher support

Documentation [🔗](#)
Certification [🔗](#)
Help [🔗](#)

Overview

Description

Publisher: 2BIT

Instead of spending countless hours on error-prone, manual data testing requiring specialized skills, 2TEST automates data and BI testing with a user-friendly interface for your entire team. Business analysts can easily define data quality expectations and test business rules without programming expertise. Developers can write unit tests and integrate automated data checks into their workflows. Release and project managers benefit from a centralized platform that streamlines testing, enhances team collaboration, and provides a comprehensive view of data quality.

License

View purchasing detail for 2TEST - Automated Data Testing and Data Quality checks (preview) on the Azure Marketplace

[Azure marketplace offer page](#) [🔗](#)

Get started

Home					
Status	Environment	Duration	Total Tests	Failed Tests	Started On
Completed successfully	Fabric DWH	12.00s	4	0	12/18/2024 4:41:12 PM
Completed successfully	Fabric DWH	12.00s	4	0	12/18/2024 4:41:12 PM
Completed successfully	Fabric DWH	12.00s	4	0	12/18/2024 4:41:12 PM



- **Informatica Cloud Data Quality**: Let's you profile, detect, and fix data issues—such as duplicates, missing values, and inconsistencies—directly within your Fabric environment.



Informatica Cloud Data Quality (preview)

Everybody's ready for AI except your data™

[Add Workload](#) [🔗](#)

About

Item types



Compatible with



Publisher support

Documentation [🔗](#)
Certification [🔗](#)
Help [🔗](#)

Overview

Description

Publisher: Informatica

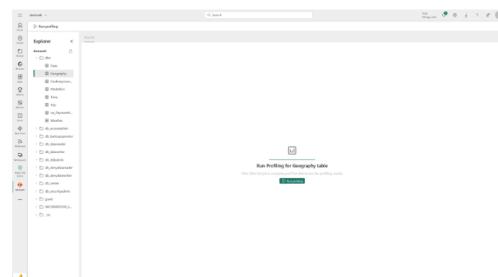
Informatica's Intelligent Data Management Cloud (IDMC) is the leading platform for cloud data management with Microsoft Fabric, offering data integration, discovery, quality, master data management, and governance. Powered by AI, CLAIRE®, IDMC connects to Fabric OneLake, Lakehouse, and Data Warehouse with 300+ connectors. The Public Preview introduces Data Quality as a native workload in Microsoft Fabric for identifying data irregularities. Users can fully leverage IDMC with Microsoft's data quality profiles for comprehensive management.

License

View purchasing detail for Informatica Cloud Data Quality (preview) on the Azure Marketplace

[Azure marketplace offer page](#) [🔗](#)

At a glance



- **Lumel EPM**: Empowers business users build no-code Enterprise Performance Management (EPM) apps on top of the semantic models.



Lumel EPM (preview)

Accelerate AI readiness by consolidating Planning, Analytics & Data Apps on Microsoft Fabric

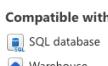
[Add Workload](#) [🔗](#)

About

Item types



Compatible with



Publisher support

Documentation [🔗](#)
Certification [🔗](#)
Help [🔗](#)

Overview

Description

Publisher: Lumel

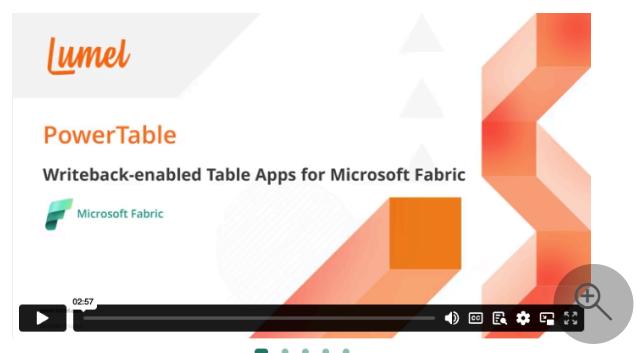
Empower your business users build no-code Enterprise Performance Management (EPM) apps on top of your semantic models. This workload unlocks three workload items: (a) Lumel Planning for prescriptive, integrated planning apps, (b) Lumel BI for descriptive analytics apps & dashboards and (c) Lumel Data Apps for acting on plans & insights. Lumel is trusted by 3,000+ customers worldwide and was voted "Best Overall Vendor" for Enterprise Performance Management in 2025 by BPM Partners.

License

View purchasing detail for Lumel EPM (preview) on the Azure Marketplace

[Azure marketplace offer page](#) [🔗](#)

At a glance



- **Neo4j AuraDB with Graph Analytics** : Create graph models from OneLake data, visually analyze and explore data connections, query your data, and run any of the 65+ built-in algorithms with a seamless experience in the Fabric Console.

The screenshot shows the product page for 'Neo4j AuraDB with graph analytics (preview)' in the Microsoft Fabric Marketplace. At the top, there's a large image of a brain inside a blue container, with a small globe and a lightbulb icon above it. Below the image, the title 'Neo4j AuraDB with graph analytics (preview)' is displayed, along with a subtext: '65+ built-in graph algorithms for use with your OneLake data, helping you uncover hidden patterns and relationships for improved decision-making.' A green 'Add Workload' button is visible. The page is divided into three main sections: 'About', 'Overview', and 'At a glance'. The 'About' section includes details like 'Item types: Neo4j Graph Dataset', 'Compatible with: Lakehouse', and 'Publisher support' links. The 'Overview' section contains a detailed description of the workload, mentioning its integration with Microsoft Fabric and its ability to help users uncover hidden patterns and relationships. It also includes a note about the public preview version and a 'License' section. The 'At a glance' section features a screenshot of the Microsoft Fabric Data Science workspace showing a graph visualization of data points and relationships.

- **Osmos AI Data Wrangler** : Automates data preparation with AI-powered data wranglers, making data transformation effortless.

The screenshot shows the product page for 'Osmos AI Data Wrangler' in the Microsoft Fabric Marketplace. At the top, there's a large image of a golden globe and abstract geometric shapes. Below the image, the title 'Osmos' and subtitle 'Accelerate your Fabric deployment with AI data agents' are displayed, along with a green 'Add Workload' button. The page is divided into three main sections: 'About', 'Overview', and 'At a glance'. The 'About' section includes details like 'Item types: AI Data Engineer, AI Data Wrangler', 'Compatible with: Lakehouse', and 'Publisher support' links. The 'Overview' section contains a detailed description of the AI Data Wrangler, stating it empowers data teams to manage their Fabric environment using AI Agents. It mentions the 'Osmos Data Engineer' for creating execution-ready Spark notebooks and the 'Osmos Data Wrangler' for wrangling messy data. The 'At a glance' section features a screenshot of a video player titled 'Accelerate Data Transformation in Microsoft Fabric... Osmos AI Agents for your data estate'. The video player shows two sections: 'AI Data Engineer' and 'AI Data Wrangler', each with a brief description and a 'Watch on YouTube' button.

- **Power Designer** : A tool for company-wide styling and report template creation, improving Power BI report designs.

tips **Power Designer**
The world's most comprehensive report design tool

Add Workload

About

Item types
[Design](#)

Publisher support
[Documentation](#) [Certification](#) [Help](#)

Overview

Description
Publisher: PowerBI.tips LLC

Power Designer enables company wide styling and creation of report templates in record time. Level up your report style by refining your report design within this novel tool.

Note: this product can be purchased from App Source and Azure Marketplace. To find this product on the stores search for "tips+" and it will be the first item found.

License

View purchasing detail for Power Designer on the Azure Marketplace
[Azure marketplace offer page](#)

At a glance

- **Celonis Process Intelligence**: Allows organizations to expose Celonis' unique class of data and context in Microsoft Fabric.

c **Celonis Process Intelligence (preview)**
Integrates Celonis' unique process data and context into Microsoft Fabric

Add Workload

About

Item types
[Data Integration](#) [Process Analysis](#)

Compatible with
[Lakehouse](#)

Publisher support
[Documentation](#) [Certification](#) [Help](#)

Overview

Description
Publisher: Celonis

With Celonis Process Intelligence for Microsoft Fabric we allow organisations to expose Celonis' unique class of data and context in Microsoft Fabric - this intelligence is the key to improving processes across systems, departments, and organizations.

License

View purchasing detail for Celonis Process Intelligence (preview) on the Azure Marketplace
[Azure marketplace offer page](#)

At a glance

- **Profisee Master Data Management**: empowers users to efficiently match, merge, standardize, remediate and validate data, transforming it into trusted, consumption-ready data products for analytics and AI.

- **Quantexa Unify**: Enhances Microsoft OneLake data sources by providing a 360-degree view with advanced data resolution capabilities.

- **SAS Decision Builder**: Helps organizations automate, optimize, and scale their decision-making processes.

SAS Decision Builder (preview)
Powering Smarter Decisions for a Data-Driven Future

Add Workload ▾

About

Item types
 Decision
 Python code file
 Rule set

Compatible with
 Lakehouse
 ML model

Publisher support
[Documentation](#)
[Certification](#)
[Help](#)

Overview

Description
Publisher: SAS Institute Inc.

SAS Decision Builder is a powerful, SaaS solution designed to help organizations automate, optimize, and scale their decision-making processes. Whether you're managing complex business rules, integrating machine learning models, or using Python code, SAS Decision Builder allows you to create intelligent decision flows that drive actionable insights in real-time. With seamless integration into the Microsoft Fabric ecosystem, SAS Decision Builder helps businesses streamline workflows, improve operational efficiency, and make smarter, data-driven decisions. Effortlessly connect to Azure services, deploy machine learning models, and monitor decision performance - all from a single platform. Start transforming your decision-making process today with SAS Decision Builder.

License

View purchasing detail for SAS Decision Builder (preview) on the Azure Marketplace
[Azure marketplace offer page](#)

At a glance

The screenshot shows a workflow titled "Loan_Request_Review". It starts with a "Start" node, followed by a "Initial_Loan_Review" node (blue), which leads to a "Loan_Status" decision node (orange). From the "Loan_Status" node, three paths emerge: "REFUSED" leading to a "Loan_Refuse_Denial" node (blue), "APPROVED" leading to a "Loan_Approved_Message" node (purple), and "DENIED" leading to a "Loan_Denied_Message" node (purple). All three message nodes converge back to an "End" node (grey).

- **Statsig:** Brings data visualization and analysis directly to your warehouse.

Statsig (preview)
Innovate with confidence

Add Workload ▾

About

Item types
 Statsig Analytics
 Statsig Experiments

Compatible with
 Warehouse

Publisher support
[Documentation](#)
[Certification](#)
[Help](#)

Overview

Description
Publisher: Statsig

Statsig brings data visualization and analysis directly to your warehouse. Analyze your data with Statsig, an industry leading product data platform.

License

View purchasing detail for Statsig (preview) on the Azure Marketplace
[Azure marketplace offer page](#)

At a glance

The screenshot shows a "Choose a data source to connect to Statsig" dialog. It lists several data sources under the "Available" tab, including "New", "Every 5", "Every 10", "Every 30", "Every 60", "Every 90", "Every 120", "Every 180", and "Every 360". Each entry has columns for "Order", "Interval", "Location", "Environment", and "Sandbox". A "Select" button is visible at the bottom right of the dialog.

- **Teradata AI Unlimited:** Combines Teradata's analytic engine with Microsoft Fabric's data management capabilities through Teradata's in-database functions.

Teradata AI Unlimited (preview)

High-performance, on-demand compute, and in-engine analytics to explore, experiment, and operationalize new use cases

Add Workload ▾



About

Item types

AI Unlimited Notebook

Publisher support

Documentation [🔗](#)

Certification [🔗](#)

Help [🔗](#)

Overview

Description

Publisher: Teradata

The AI Unlimited workload combines Teradata's analytic engine with Microsoft Fabric's comprehensive data management and lifecycle applications. It is now easier than ever for data scientists and data engineers to rapidly experiment, prototype, and iterate using Teradata's in-database functions.

Note: This preview version of AI Unlimited is hosted in the US only. AI Unlimited will not store data but, by accepting this Workload, you will allow users in your organization to send data to the **US East** region for processing. If you are outside of the US, please consult your company's data policy before accepting the AI Unlimited Workload.

License

View purchasing detail for Teradata AI Unlimited (preview) on the Azure Marketplace

[Azure marketplace offer page](#) [🔗](#)

At a glance

The screenshot shows a Microsoft Fabric interface for a 'ml_pipeline' workload. The main area displays a query editor with T-SQL code for training a GLM model. The code includes a SELECT statement with a PARTITION BY ANY clause and a CREATE MODEL statement. Below the code, there are sections for 'Model training' and 'Model evaluation'. A preview pane shows the results of the query, which is a table of various metrics related to the training process. The interface includes standard Microsoft Fabric navigation and search tools.



- **SQL2Fabric-Mirroring by Striim** [🔗](#): Fully managed, zero-code replication solution that seamlessly mirrors on-premises SQL Server data to Microsoft Fabric OneLake

SQL2Fabric-Mirroring (preview)

SQL2Fabric Mirroring: Seamless, Real-Time Data Integration to Microsoft Fabric

Add Workload ▾



About

Overview

At a glance

Item types

SQL2Fabric-Mirroring

Compatible with

Lakehouse

Publisher support

Documentation [🔗](#)

Certification [🔗](#)

Help [🔗](#)

Description

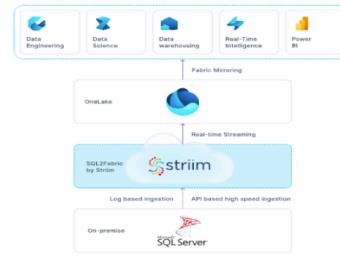
Publisher: Striim, Inc.

Striim's SQL2Fabric-Mirroring is a fully managed, zero-code replication solution that seamlessly mirrors on-premises SQL Server data to Microsoft Fabric OneLake. Powered by Striim's high-performance, secure, and scalable cloud platform, this low cost and low latency service delivers on-premise transactional data into Fabric for AI, BI, Analytics, Data Engineering, and Operational Reporting use cases.

License

View purchasing detail for SQL2Fabric-Mirroring (preview) on the Azure Marketplace

[Azure marketplace offer page](#) [🔗](#)



As more workloads become available, the Workload Hub will continue to serve as a dynamic space for discovering new capabilities, ensuring that users have the tools they need to scale and optimize their data-driven solutions.

Last updated on 12/15/2025