

Power BI Desktop developer mode documentation

Microsoft Power BI Desktop developer mode brings Pro BI developer experiences right into Power BI Desktop. It provides a suite of features that enable developer focused capabilities like external tool support, script and API support, source control with Git integration, and Continuous Integration/Continuous Delivery (CI\CD) with Azure DevOps.

Power BI Desktop projects

CONCEPT

[Power BI Desktop projects overview](#)

[Project semantic model folder](#)

[Project report folder](#)

[Project Git integration](#)

[Project Azure DevOps integration](#)

External tools

CONCEPT

[External tools in Power BI Desktop](#)

[Register an external tool](#)

Power BI Desktop projects (PREVIEW)

Article • 08/25/2023

Important

Power BI Desktop projects is currently in **preview**.

Power BI Desktop introduces a new way to author, collaborate, and save your projects. You can now save your work as a **Power BI Project** (PBIP). As a project, report and dataset *artifact* definitions are saved as individual plain text files in a simple, intuitive folder structure.

Saving your work as a project has the following benefits:

- **Text editor support** - Artifact definition files are JSON formatted text files containing model dataset and report metadata. They're publicly documented and human readable. While project files support simple text editing tools like Notepad, it's better to use a code editor like [Visual Studio Code \(VS Code\)](#), which provides a rich editing experience including intellisense, validation, and Git integration.
- **Programmatic generation and editing artifact definitions** - You can create scripts using the popular and easy to use [Tabular Model Scripting Language \(TMSL\)](#), or create your own custom applications to make changes to your artifact definitions. Applications can be based on public documentation of the artifact definition schemas and/or client libraries.
- **Source control** - Power BI dataset and report artifact definitions can be stored in a source control system, like Git. With Git, you can track version history, compare revisions (diff), and revert to previous versions. Source control can also unblock collaboration when using Power BI Desktop by using familiar collaboration mechanisms for resolving conflicts (merge) and reviewing changes (pull requests). To learn more, see [Version control in Git](#).
- **Continuous Integration and Continuous Delivery (CI/CD)** - You can use systems where developers in your organization submit a proposed change to the CI/CD system. The system then validates the change with a series of *quality gates* before applying the change to the production system. These quality gates can include code reviews by other developers, automated testing, and automated build to validate the integrity of the changes. CI/CD systems are typically built on top of existing source control systems. To learn more, see [DevOps - Continuous integration](#), and [DevOps - Continuous delivery](#).

Video

See Power BI Desktop projects and other developer mode features being introduced at Microsoft Build 2023.

<https://www.youtube-nocookie.com/embed/OdkS7DF7EIY?start=277>

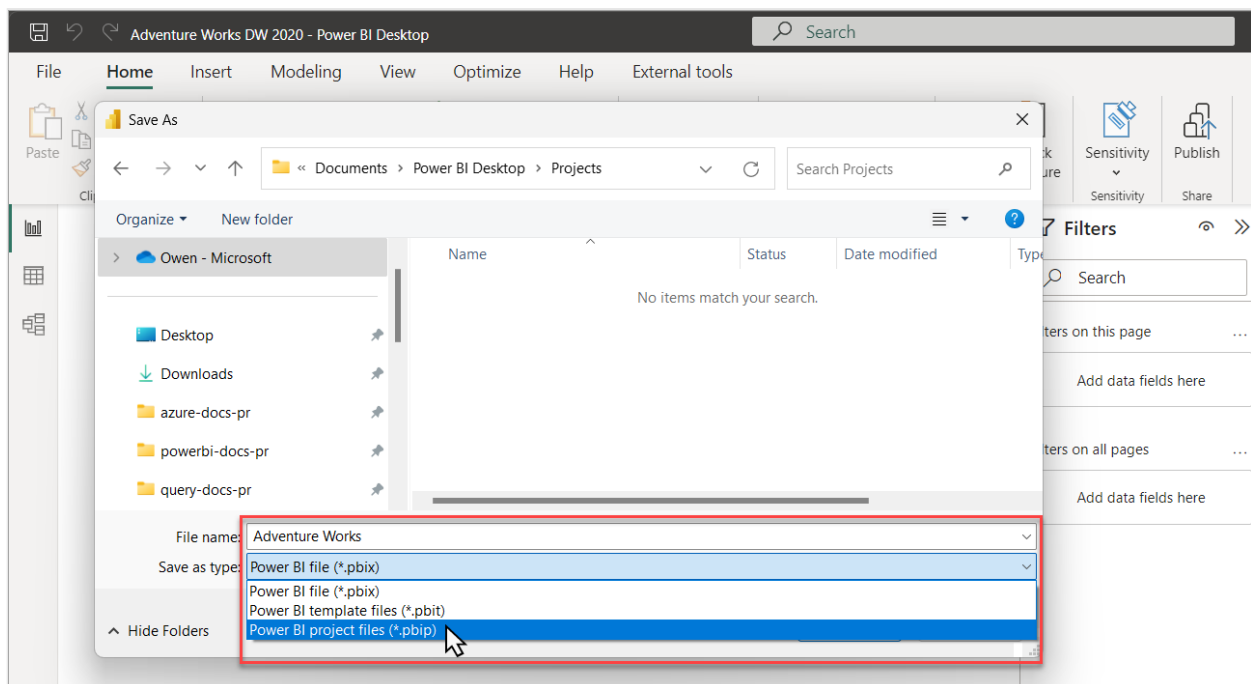
Enable preview features

Saving as a project in Power BI Desktop is currently in **preview**. Before giving it a try, you must first enable it in **Preview features**.

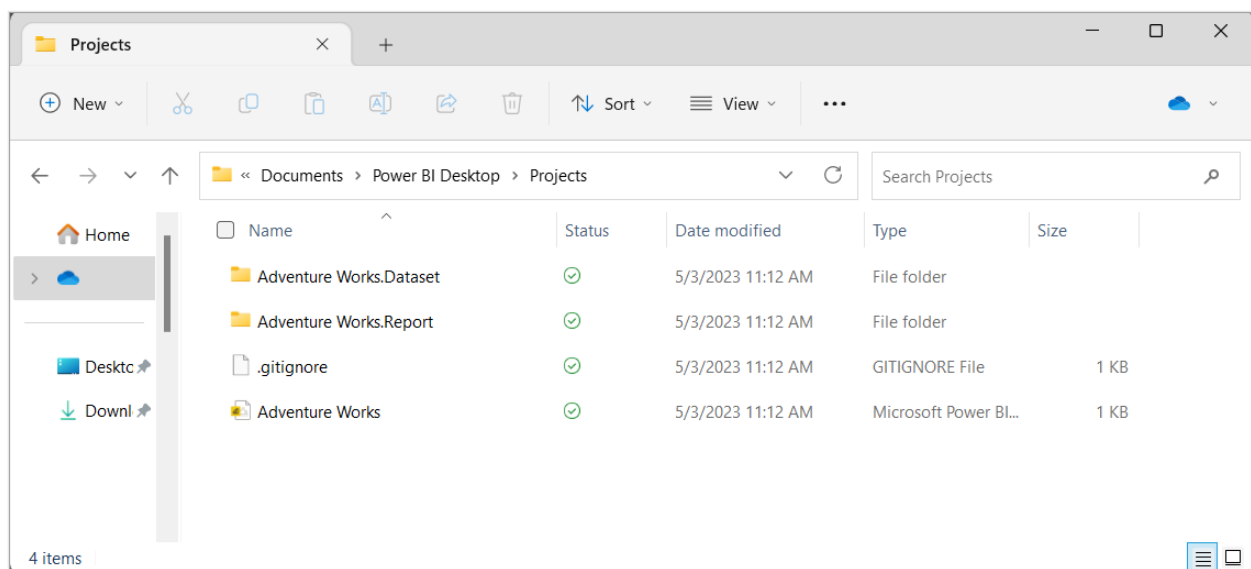
To enable, in Power BI Desktop > **File** > **Options and settings** > **Options** > **Preview features**, select the checkbox for **Power BI Project (.pbip)** save option.

Save as a project

If you're working on a new project or you've opened an existing Power BI Desktop file (pbix), you can save your work as a Power BI *project* file (pbip):



When you save as a project, Power BI Desktop saves report and dataset artifacts as folders, each containing text files that define the artifact. You see the following:



Let's take a closer look at what you see in your project's root folder:

<project name>.Dataset

A collection of files and folders that represent a Power BI dataset. It contains some of the most important files you're likely to work on, like `model.bim`. To learn more about the files and subfolders and files in here, see [Project Dataset folder](#).

<project name>.Report

A collection of files and folders that represent a Power BI report. To learn more about the files and subfolders and files in here, see [Project report folder](#).

.gitignore

Specifies intentionally untracked files Git should ignore. Power BI Desktop creates the [.gitignore](#) file in the root folder when saving if it doesn't already exist.

Dataset and report subfolders each have default git ignored files specified in `.gitignore`:

- Dataset
 - `.pbi\localSettings.json`
 - `.pbi\cache.abf`
- Report
 - `.pbi\localSettings.json`

<project name>.pbip

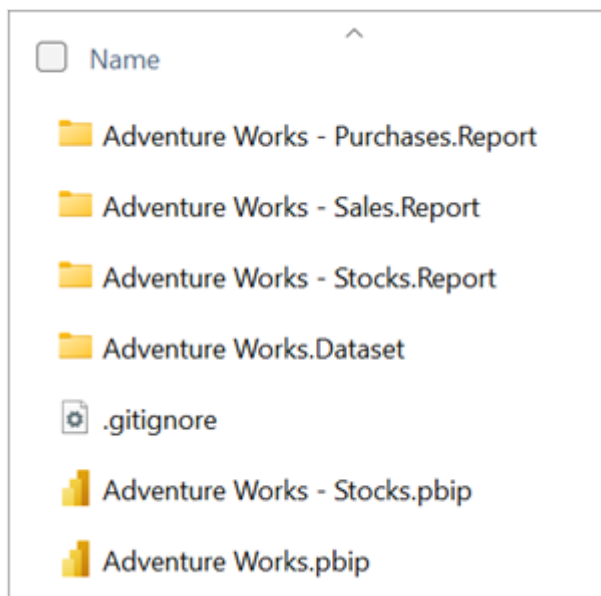
The PBIP file contains a pointer to a report folder, opening a PBIP opens the targeted report and model for authoring.

For more information, refer to the [pbip schema document](#).

Open a Power BI Project

You can open Power BI Desktop from the Power BI Project folder either by opening the **pbip** file or the **pbir** file in the report folder. Both options open the report for editing, and the dataset, if there's a relative reference to a dataset.

You can save multiple reports and datasets to the same folder. Having a separate pbip file for each report isn't required because you can open each report directly from the pbir within the report folder.

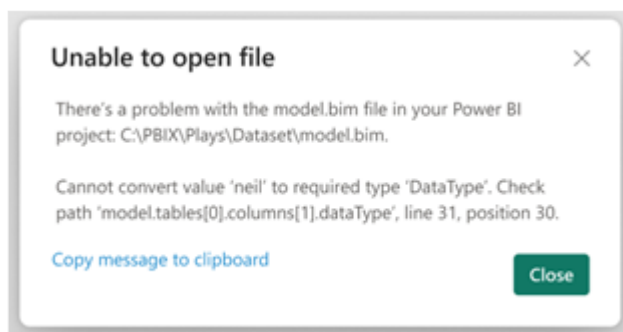


Changes outside Power BI Desktop

When saved as a project, you're not forced into making changes to your dataset and report definitions only in Power BI Desktop. You can use other tools such as VS Code, open-source community tools like Tabular Editor, or even Notepad. However, not every file or change supports editing by external, open-source tools.

Changes to files or properties outside of Power BI Desktop can cause unexpected errors, or even prevent Power BI Desktop from opening. In those cases, you must resolve the issues in the files before trying to open the project again in Power BI Desktop.

If possible, Power BI Desktop indicates the file and location of error:



Schema details for the following files aren't documented. During **preview**, changes to these files outside of Power BI Desktop aren't supported:

- Report\
 - [report.json](#)
 - [mobileState.json](#)
 - [datasetDiagramLayout.json](#)
- Dataset\
 - [diagramLayout.json](#)

Model authoring

You can make changes to the model definition by using external tools in two ways:

- By connecting to Power BI Desktop's Analysis Service (AS) instance with [external tools](#).
- By editing JSON metadata in the model.bim file using VS Code or another external tool.

Not every model object supports write operations. Applying changes outside of the those supported can cause unexpected results.

Objects that support write operations:

| Object | Connect to AS instance | File change |
|--------------------|---------------------------------|-------------|
| Tables | No | Yes |
| Columns | Yes ¹ , ² | Yes |
| Calculated tables | Yes | Yes |
| Calculated columns | Yes | Yes |
| Hierarchies | Yes | Yes |
| Relationships | Yes | Yes |

| Object | Connect to AS instance | File change |
|-----------------------------|------------------------|---------------------------------|
| Measures | Yes | Yes |
| Model KPIs | Yes | Yes |
| Calculation groups | Yes | Yes |
| Perspectives | Yes | Yes |
| Translations | Yes | Yes |
| Row Level Security (RLS) | Yes | Yes |
| Object Level Security (OLS) | Yes | Yes |
| Annotations | Yes | Yes |
| M expressions | No | Yes ³ , ⁴ |

Keep in mind:

- Any changes to open files made outside Power BI Desktop requires a restart for those changes to be shown in Power BI Desktop. Power BI Desktop isn't aware of changes to project files made by other tools.
- Power BI Desktop doesn't support tables with multiple partitions. Only a single partition for each table is supported. Creating tables with empty partitions or more than one partition results in an error when opening the report.
- Automatic date tables created by Power BI Desktop shouldn't be changed by using external tools.
- When changing a model that uses Direct Query to connect a Power BI dataset or Analysis Services model, you must update the ChangedProperties and PBI_RemovedChildren collection for the changed object to include any modified or removed properties. If ChangedProperties and/or PBI_RemovedChildren isn't updated, Power BI Desktop may overwrite any changes the next time the query is edited or the model is refreshed in Power BI Desktop.
- ¹ - Changing a column's data type is supported. However, renaming columns isn't supported when connecting to the AS instance.
- ² - If the dataset has the [Auto date/time](#) feature enabled, and you create a new datetime column outside of Power BI Desktop, the local date table isn't automatically generated.

- 3 - Partition [SourceType](#) must be Calculated, M, Entity, or CalculationGroup. Partition [Mode](#) must be Import, DirectQuery, or Dual.
- 4 - Any expression edits outside of Power BI Desktop in a project with [unappliedChanges.json](#) are lost when those changes are applied.

JSON file schemas

Most project files contain metadata in JSON format. Corresponding JSON schemas can be used for validation and documentation.

With JSON schemas, you can:

- Learn about configurable properties.
- Use inline JSON validation provided by the code editor.
- Improve authoring with syntax highlighting, tooltips, and autocomplete.
- Use external tools with knowledge of supported properties within project metadata.

Use VS Code to map JSON schemas to the files being authored. JSON schemas for project files are provided in the [Power BI Desktop samples Git repo](#).

Considerations and limitations

- Power BI Desktop isn't aware of changes made with other tools or applications. Changes made by using external tools require you to restart Power BI Desktop before those changes are shown.
- Sensitivity labels aren't supported with Power BI projects.
- Download PBIX isn't supported for workspaces with Git integration.
- Diagram view is ignored when editing models in the Service.
- When saving as a Power BI Project, the maximum length of the project files path is 260 characters.
- In Power BI Desktop, you can't save as a PBIP directly to OneDrive for Business and SharePoint.
- When editing PBIP files outside of Power BI Desktop, they should be saved using UTF-8 without BOM encoding.
- Report Linguistic Schema is not supported with Power BI projects.
- Power BI Desktop uses CRLF as end-of-line. To avoid problems in your diffs, configure Git to handle line endings by enabling [autocrlf](#).

Frequently asked questions

Question: Looking at dataset and report artifact folder definitions only a few files are marked as required, what happens if I delete them?

Answer: Power BI Desktop automatically creates them when you save as a project (PBIP).

Question: Is Power BI Desktop aware of changes I make to the Power BI Project files from an external tool or application?

Answer: No. Any change made to the files requires Power BI Desktop to be restarted to reflect those changes.

Question: If I convert a PBIX to a PBIP, can I convert it back to PBIX?

Answer: Yes. You can save a PBIX as a PBIP, or save a PBIP as a PBIX.

Question: Can I convert PBIX into PBIP and vice-versa programmatically?

Answer: No. You can only convert a PBIX into a PBIP and vice-versa using Power BI Desktop's **File > Save as**.

Question: Can I deploy a Power BI Desktop project to Azure Analysis Services (AAS) or SQL Server Analysis Services (SSAS)?

Answer: No. Power BI Desktop project report definitions aren't supported in AAS and SSAS. And model definitions use an enhanced metadata unique to Power BI. For AAS and SSAS projects, use Microsoft Visual Studio for model authoring, Git, and Azure DevOps integration.

See also

[Power BI Desktop project dataset folder](#)

[Power BI Desktop project report folder](#)

[Power BI Desktop projects Git integration](#)

[Power BI Desktop projects Azure DevOps integration](#)

[External tools in Power BI Desktop](#)

Power BI Desktop project dataset folder

Article • 07/28/2023

Important

Power BI Desktop projects is currently in **preview**.

This article describes the files and subfolders in a Microsoft Power BI Desktop project's **Dataset** folder. The files and subfolders here represent a Power BI dataset. Depending on your project, the dataset folder can include:

- .pbi\
 - [localSettings.json](#)
 - [editorSettings.json](#)
 - [cache.abf](#)
 - [unappliedChanges.json](#)
- [model.bim](#)¹
- [definition.pbidataset](#)¹
- [diagramLayout.json](#)
- [item.config.json](#)
- [item.metadata.json](#)

¹ - This file is required.

Not every project dataset folder includes all of the files and subfolders described here.

Dataset files

.pbi\localSettings.json

Contains dataset settings that apply only for the current user and computer. It should be included in gitignore or other source control exclusions. By default, this file is ignored by Git.

For more information, refer to the [localSettings.json schema document](#) .

.pbi\editorSettings.json

Contains dataset editor settings saved as part of the dataset definition for use across users and environments.

For more information, refer to the [editorSettings.json schema document](#).

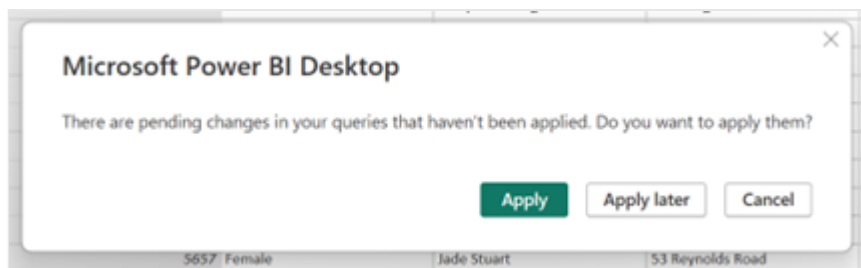
.pbi\cache.abf

An Analysis Services Backup (ABF) file containing a local cached copy of the model and data when it was last edited. It should be included in gitignore or other source control exclusions. By default, this file is ignored by Git.

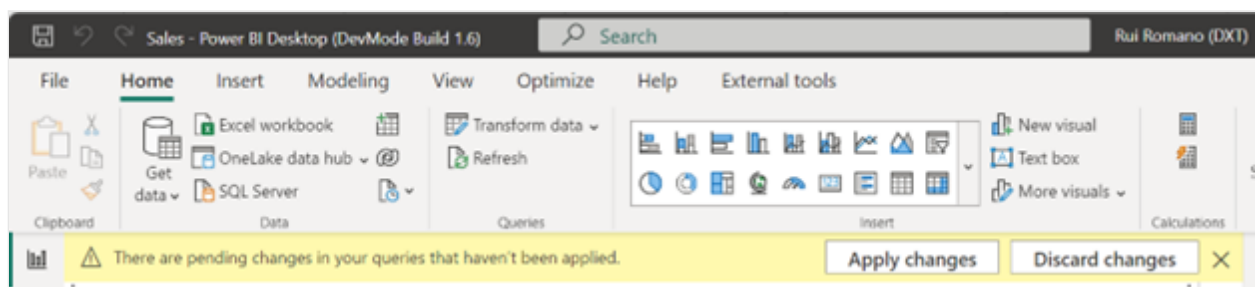
Power BI Desktop can open a project without a cache.abf file. In that case, it opens the report connected to a model with its entire definition but without data. If a cache.abf exists, Power BI Desktop loads the data and overwrites the model definition with the content in model.bim.

.pbi\unappliedChanges.json

Power BI Desktop allows you to save changes made in the Transform Data editor (Power Query) without first applying those changes to the data model.



When you select **Apply later**, the unapplied changes are saved into the unappliedChanges.json file. When pending changes are in the unappliedChanges file, Power BI Desktop prompts you to apply or discard those pending changes:



If you select **Apply changes**, Power BI Desktop overwrites the queries in model.bim with the queries from unappliedChanges.json. If you edited queries in model.bim outside of Power BI Desktop and there is a previous unappliedChanges.json file, your changes are lost and replaced by the queries in unappliedChanges.json when those changes are applied.

The `unappliedChanges.json` file is automatically incorporated into the dataset definition and saved in Git by default. This allows you to commit your ongoing work to the development branch, serving as a backup and making it accessible to other team members. However, you have the option to exclude this file from Git's tracking, preventing unfinished query work from affecting other developers.

For more information, refer to the [unappliedChanges.json schema document](#).

model.bim

Contains a Tabular Model Scripting Language (TMSL) [Database object](#) definition of the project model.

definition.pbidataset

Contains the overall definition of a dataset and core settings.

For more information, refer to the [definition.pbidataset schema document](#).

diagramLayout.json

Contains diagram metadata that defines the structure of the dataset associated with the report. During **PREVIEW**, this file doesn't support external editing.

item.config.json

Identifies the folder as a source control representation of a service item. To learn more, see [Git integration source code format - Config file](#).

For more information, refer to the [item.config.json schema document](#).

item.metadata.json

Contains attributes that define the item. To learn more, see [Git integration source code format - Metadata file](#)

For more information, refer to the [item.metadata.json schema document](#).

See also

Power BI Desktop project report folder

Power BI Desktop projects

Tabular Model Scripting Language (TMSL)

Power BI Desktop project report folder

Article • 07/28/2023

Important

Power BI Desktop projects is currently in **preview**.

This article describes the files and subfolders in a Microsoft Power BI Desktop project's **Report** folder. The files and subfolders here represent a Power BI report. Depending on your project, the report folder can include:

- .pbi\
 - [localSettings.json](#)
- CustomVisuals\
 - [StaticResources\RegisteredResources\](#)
- [datasetDiagramLayout.json](#)
- [definition.pbir](#)¹
- [mobileState.json](#)
- [report.json](#)¹
- [item.config.json](#)
- [item.metadata.json](#)

¹ - This file is required.

Not every project report folder includes all of the files and subfolders described here.

Report files

.pbi\localSettings.json

Contains report settings that apply only for the current user and local computer. It should be included in gitignore or other source control exclusions. By default, Git ignores this file.

For more information, refer to the [localSettings.json schema document](#) .

CustomVisuals\

A subfolder that contains metadata for custom visuals in the report. Power BI supports three kinds of custom visuals:

- Organizational store visuals - Organizations can approve and deploy custom visuals to Power BI for their organization. To learn more, see [Organization store](#).
- AppSource Power BI visuals - Also known as "Public custom visuals". These visuals are available from Microsoft AppSource. Report developers can install these visuals directly from Power BI Desktop.
- Custom visual files - Also known as "Private custom visuals". The files can be loaded into the report by uploading a pbviz package.

Only private custom visuals are loaded into the CustomVisuals folder. AppSource and Organization visuals are loaded automatically by Power BI Desktop.

RegisteredResources\

A subfolder that includes resource files specific to the report and loaded by the user, like custom themes, images, and custom visuals (pbviz files).

Developers are responsible for the files here and changes are supported. For example, you can change a file and after a Power BI Desktop restart, the new file is loaded into the report. This folder can unblock some useful scenarios, like:

- Authoring custom themes outside of Power BI Desktop by using the public schema.
- Applying batch changes by changing the resource file on multiple reports. For example, you can switch the corporate custom theme, change between light and dark themes, and change logo images.

Every resource file must have a corresponding entry in the report.json file, which during **preview** doesn't support editing. Edits to RegisteredResources files are only supported for already loaded resources that cause Power BI Desktop to register the resource in report.json.

datasetDiagramLayout.json

Contains data model diagrams describing the structure of the dataset associated with the report. During **preview**, this file doesn't support external editing.

definition.pbir

Contains the overall definition of a report and core settings. This file also holds the reference to the dataset used by the report. Power BI Desktop can open a pbir file directly, just the same as if the report were opened from a pbip file. Opening a pbir also opens the dataset alongside if there's a relative reference using `byPath`.

Example definition.pbir:

JSON

```
{
  "version": "1.0",
  "datasetReference": {
    "byPath": {
      "path": "../Sales.Dataset"
    },
    "byConnection": null
  }
}
```

The definition includes the `datasetReference` property, which references the dataset used in the report. The reference can be either:

`byPath` - Specifies a relative path to the target dataset folder. Absolute paths aren't supported. A backslash (\) is used as a folder separator. When used, Power BI Desktop also opens the dataset in full edit mode.

`byConnection` - Specifies a remote dataset in the Power BI service by using a connection string. When a `byConnection` reference is used, Power BI Desktop doesn't open the dataset in edit mode.

When using a `byConnection` reference, the following properties must be specified:

| Property | Description |
|---------------------------|---|
| connectionString | The connection string referring to the remote dataset. |
| pbiModelDatabaseName | The remote dataset ID. |
| connectionType | Type of connection. For service remote dataset, this value should be <code>pbiServiceXmlaStyleLive</code> . |
| pbiModelVirtualServerName | An internal property that should have the value, <code>sobe_wowvirtualserver</code> . |

Example using `byConnection`:

JSON

```
{
  "version": "1.0",
  "datasetReference": {
    "byPath": null,
    "byConnection": {
      "connectionString": "Data
Source=\"powerbi://api.powerbi.com/v1.0/myorg/Datasets\";Initial
Catalog=Sales;Integrated Security=ClaimsToken",
      "pbiServiceModelId": null,
      "pbiModelVirtualServerName": "sobe_wowvirtualserver",
      "pbiModelDatabaseName": "e244efd3-e253-4390-be28-6be45d9da47e",
      "connectionType": "pbiServiceXmlaStyleLive",
      "name": null
    }
  }
}
```

For more information, refer to the [definition.pbir schema document](#).

mobileState.json

Contains report appearance and behavior settings when rendering on a mobile device. This file doesn't support external editing.

report.json

Defines a report including visuals, page layout, and intended interactions. During preview, this file doesn't support external editing.

item.config.json

Identifies the folder as a source control representation of a service item. To learn more, see [Git integration source code format - Config file](#).

For more information, refer to the [item.config.json schema document](#).

item.metadata.json

Contains attributes that define the item. To learn more, see [Git integration source code format - Metadata file](#)

For more information, refer to the [item.metadata.json schema document](#).

See also

[Power BI Desktop project dataset folder](#)

[Power BI Desktop projects](#)

Power BI Desktop projects Git integration

Article • 07/28/2023

Important

Power BI Desktop projects is currently in **preview**.

Git integration in Microsoft Visual Studio Code (VS Code) enables Pro BI developers working with Power BI Desktop projects to streamline development processes, source control, and collaboration with Git repositories.

With Git integration, you can:

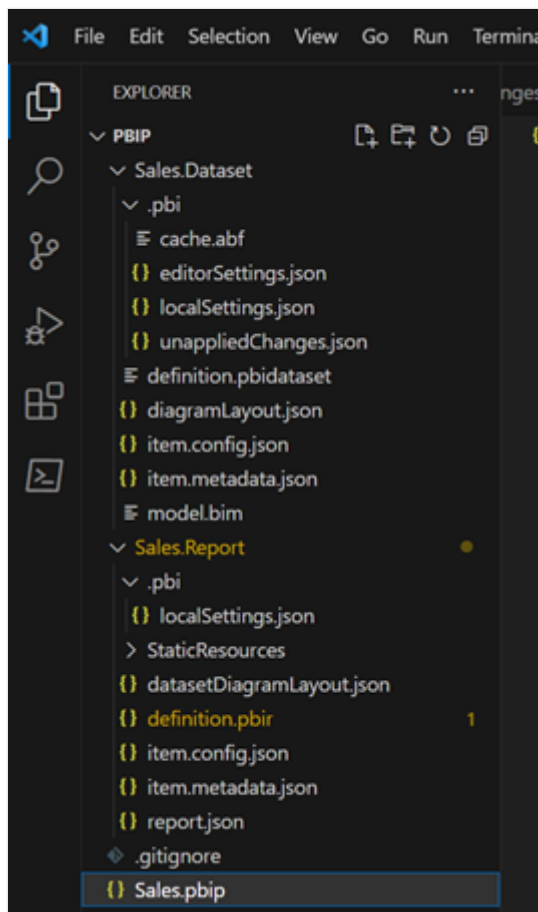
- Backup and version your work.
- Revert to previous states.
- Collaborate with others or work alone using Git branches.
- Use the capabilities of familiar source control tools, like Azure DevOps.

Prerequisites

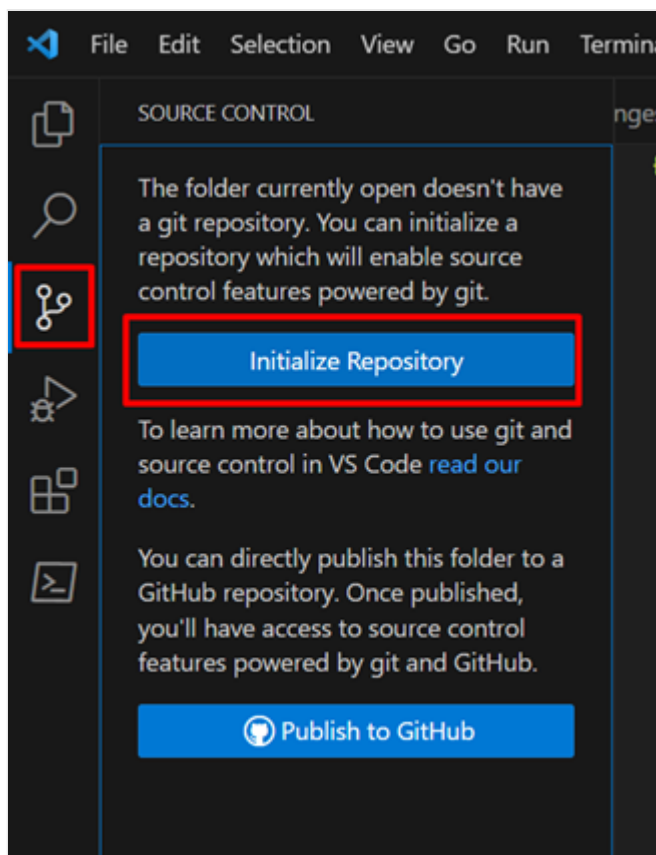
- Be familiar with Git. See [Git and GitHub learning resources](#).
- [Download](#) and install Git.
- [Download](#) and install VS Code development environment. It has native integration with Git. To learn more, see [Using Git source control in VS Code](#).

Create a local Git repo using VS Code

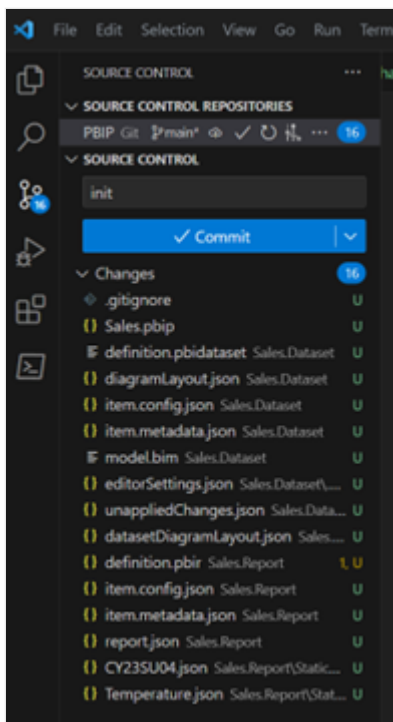
1. In VS Code, open a Power BI Desktop project folder:



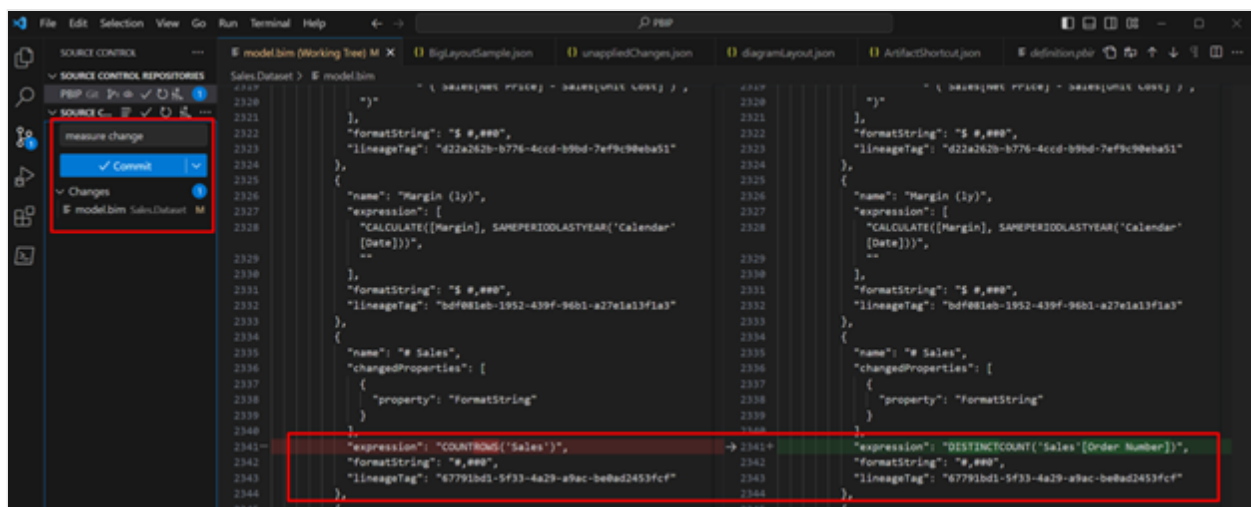
2. Initialize a Git repository by selecting **Source Control** > **Initialize Repository**:



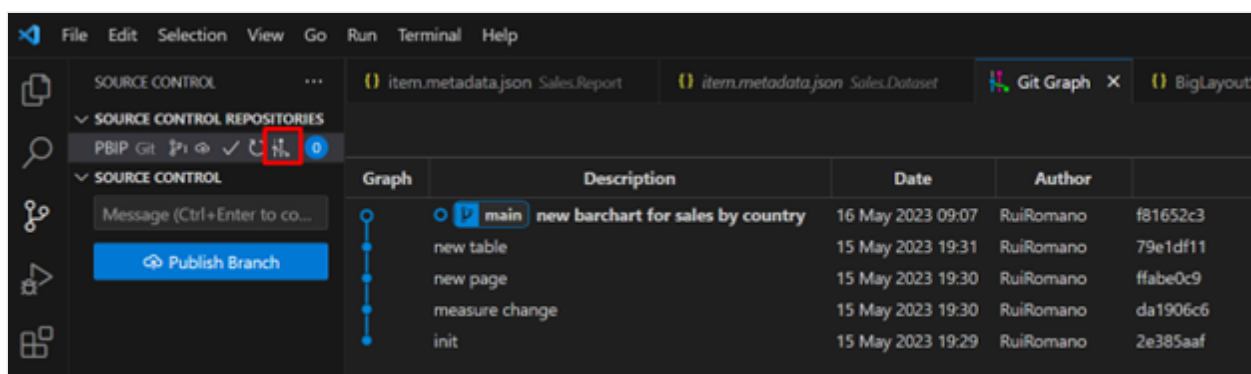
3. Do an initial Commit and enter a message:



From now on, any changes you make in Power BI Desktop changes a file in the folder tracked by your local Git. For example, in Power BI Desktop, when you change a DAX formula for a measure and then save, it triggers a Git diff on the model.bim file.



With Git integration, you can not only backup your work, but also track your change history. For example, with GitGraph, a popular free VS Code extension, you can easily track all your changes.



See also

[Power BI Desktop projects Azure DevOps integration](#)

[Power BI Desktop project dataset folder](#)

[Power BI Desktop project report folder](#)

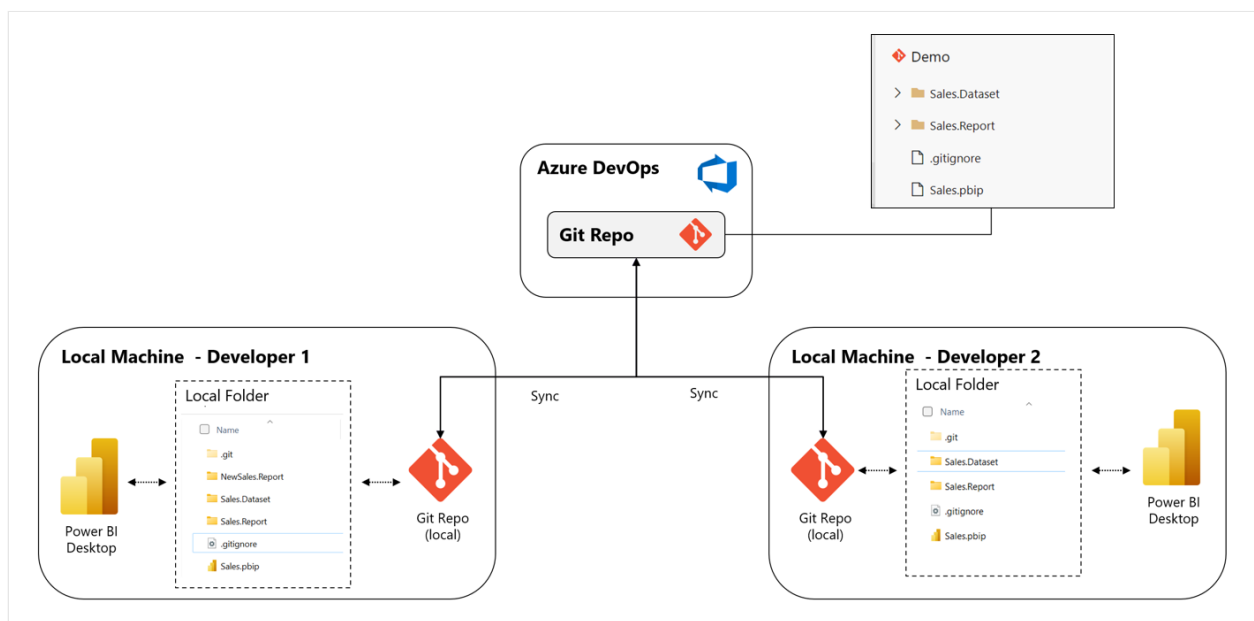
Power BI Desktop projects Azure DevOps integration

Article • 07/28/2023

Important

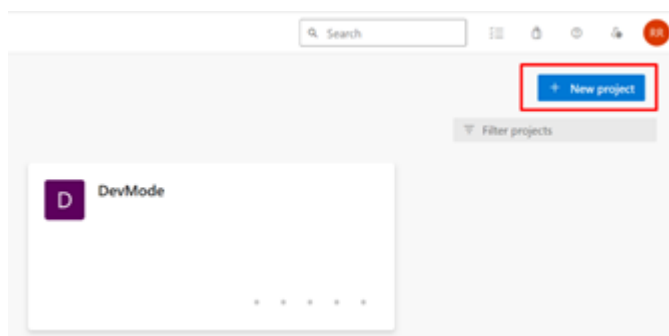
Power BI Desktop projects is currently in **preview**.

Microsoft Power BI Desktop projects enable developer collaboration by connecting your local Git repo to a remote Git host, like Azure DevOps.



Create a Git repo in Azure DevOps

1. In [Azure DevOps](#), select an existing organization, or create a new one.
2. Create a new Project within the organization:



3. Enter your project details.

Create new project

Project name *
TestProject

Description

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

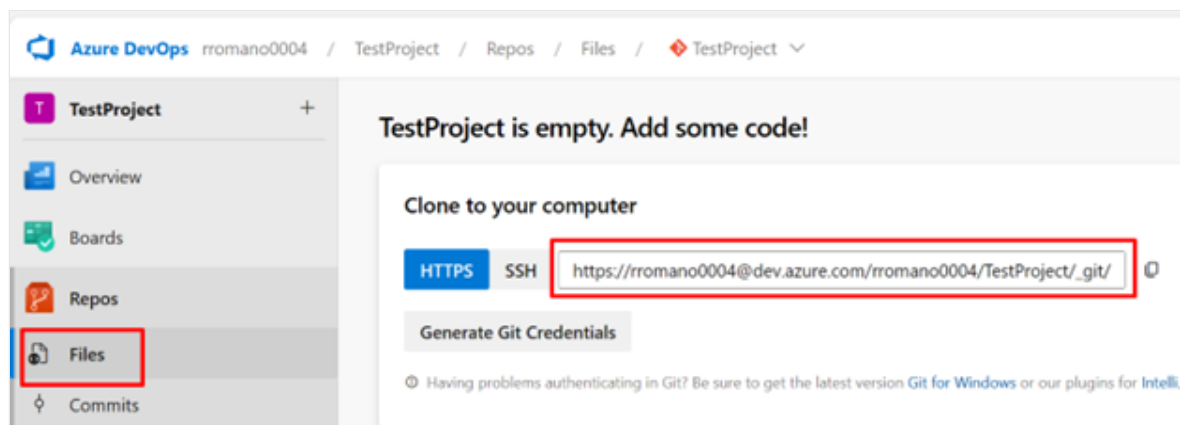
Enterprise
Members of your enterprise can view the project.

Private
Only people you give access to will be able to view this project.

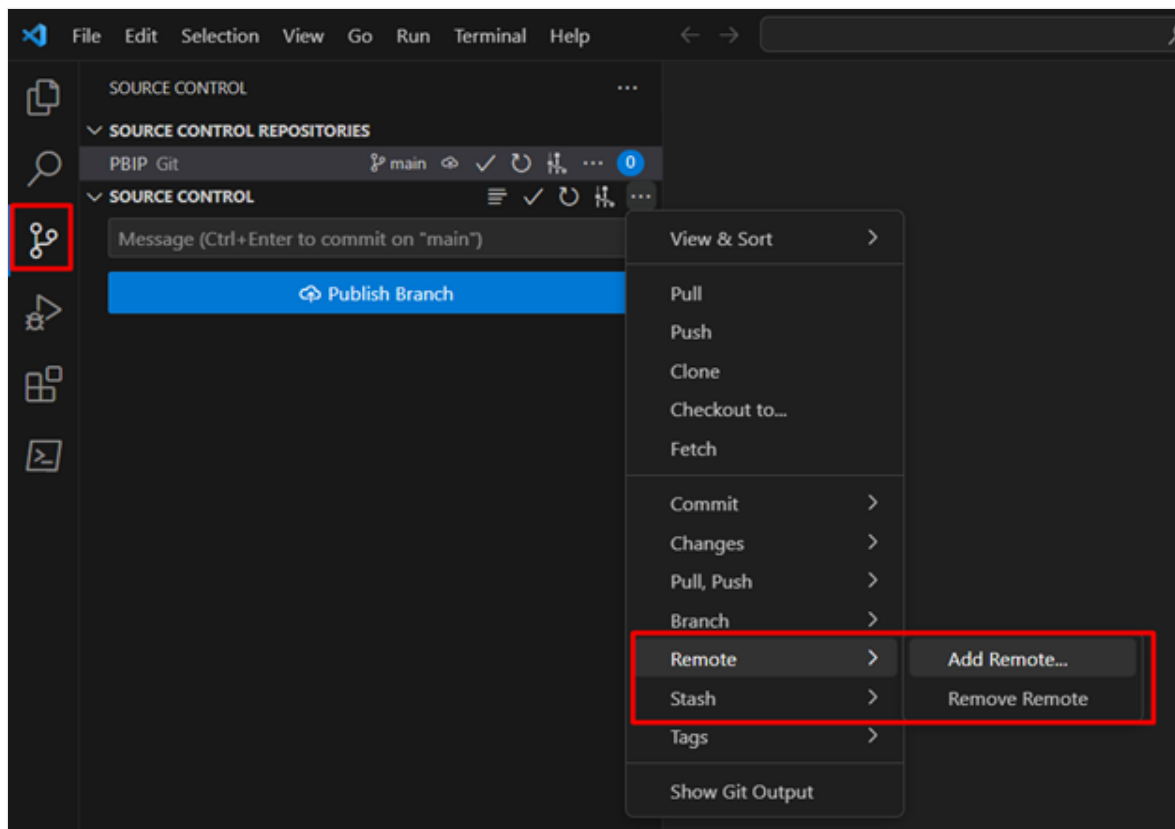
Public projects are disabled for your organization. You can turn on public visibility with organization policies.

Advanced

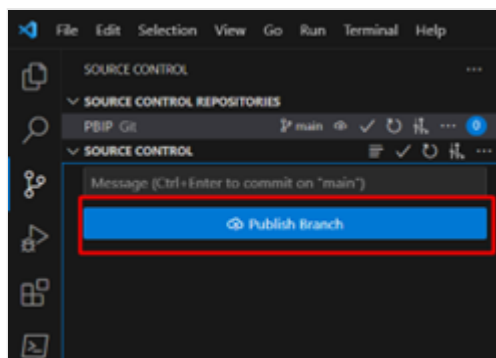
4. Select **Repos > Files**, and then copy the URL of the remote repo:



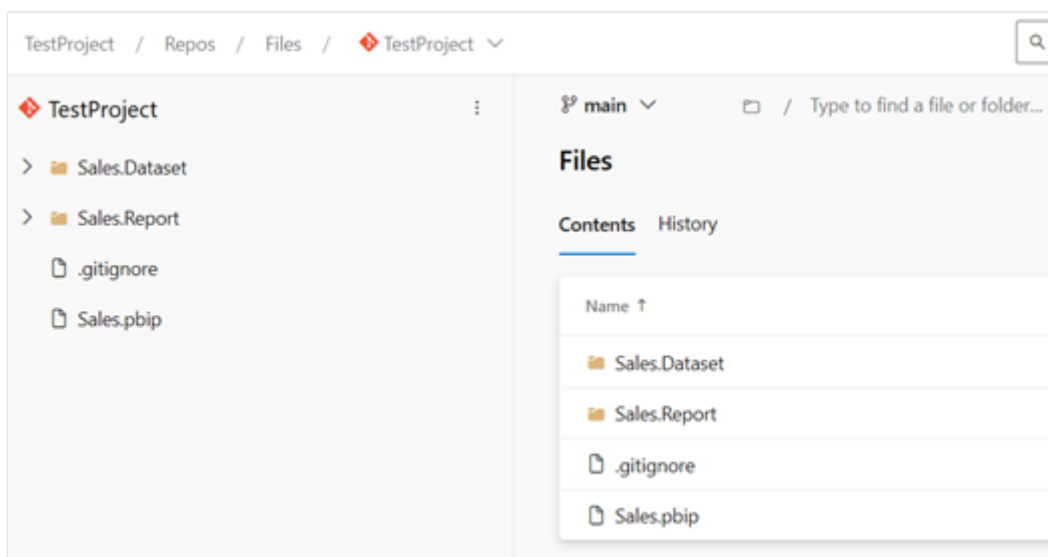
5. In Visual Studio Code (VS Code) > **Source Control > Remote**, select **Add Remote**.



6. Select **Publish Branch**.



VS Code takes care of publishing your project into Azure DevOps, where you can see your project files.



And that's it! You can see with Azure DevOps integration, you can now have multiple developers working on the same Power BI project. All they need to do is be synced with the same Azure Devops Git Repo.

If you're using Microsoft Fabric, you can also connect a Fabric workspace to an Azure DevOps Git repo and get all your content automatically deployed into the service. Git and Azure DevOps integration can provide a continuous integration workflow not only from Power BI Desktop to the service, but also from changes made in the service to Power BI Desktop. To learn more, see [Microsoft Fabric - Introduction to git integration](#).

See also

[Power BI Desktop projects Git integration](#)

[Power BI Desktop projects](#)

Power BI Project (PBIP) and Azure DevOps build pipelines for continuous integration

Article • 11/10/2023

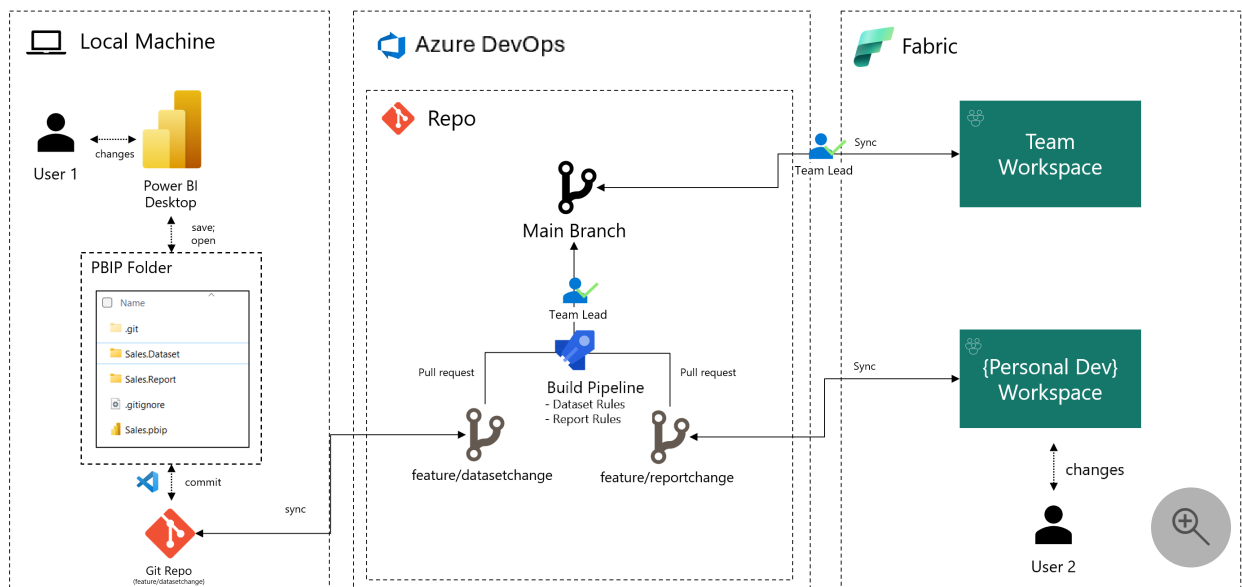
Combining Fabric Git Integration with Azure DevOps, enables you to connect a workspace to a branch in an Azure DevOps repository and automatically synchronizes between them.

Integrating the PBIP format with Azure DevOps lets you use Azure Pipelines to automate [Continuous Integration/Continuous Deployment \(CI/CD\)](#) pipelines. These pipelines process the PBIP metadata files and apply a series of quality checks to your development before deploying it to the production system.

In this article, we focus on continuous integration and describe how to create an Azure DevOps build pipeline that guarantees best practices for all semantic models and reports within a Fabric workspace. By implementing automated quality tests, you can prevent common mistakes, and enhances team efficiency. For example, this approach ensures that new team members adhere to established standards for semantic model and report development.

Learn more about PBIP and Fabric Git Integration in [project-overview](#) and [Fabric Git integration overview](#).

The following diagram illustrates the end-to-end scenario with two development workflows that trigger the Azure DevOps build pipeline to validate development quality. The build execute does the following actions:



1. *User 1* develops [using Power BI Desktop](#).
 - a. Create a branch from main using **VS Code** (feature/datasetchange)
 - b. Make changes to semantic model using Power BI Desktop
 - c. Commit changes to remote repository branch using **VS Code**
 - d. Create Pull Request to main branch using Azure DevOps
2. At the same time, *User 2* develops [using another Fabric workspace](#).
 - a. Create branch from main using Fabric Git (feature/reportchange)
 - b. Make report changes in the Fabric workspace
 - c. Commit changes to remote repository branch using Fabric Git
 - d. Create Pull Request to main branch using Azure DevOps
3. The team lead reviews the Pull Requests and synchronizes the changes to the team workspace using Fabric Git.
4. The Pull Request triggers the Azure DevOps build pipeline to inspect the semantic model and report development quality.

ⓘ Note

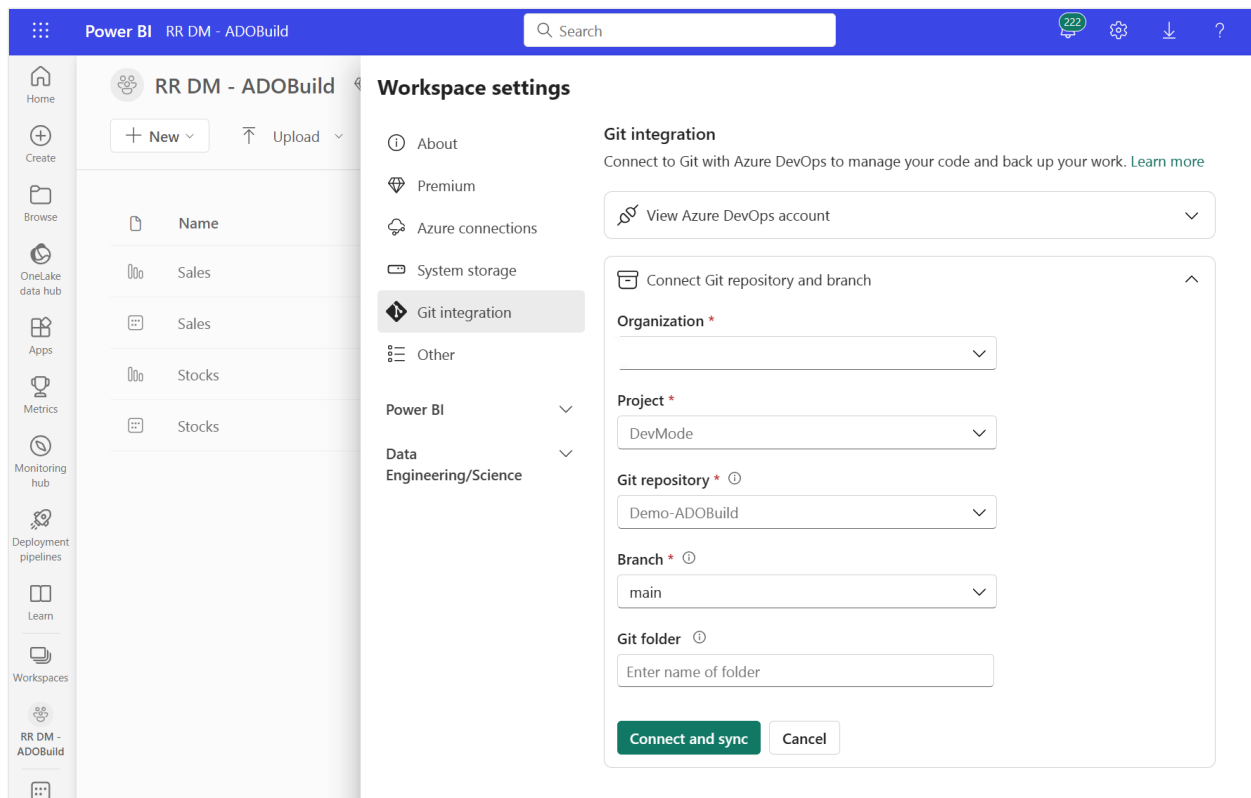
In this example, the build pipeline uses two open-source community tools that enable a developer to apply (customizable) best practice rules to the metadata of semantic models and reports within a Power BI Project folder:

- **Tabular Editor** [↗](#) and **Best Practice Rules** [↗](#)
- **PBI Inspector** [↗](#)

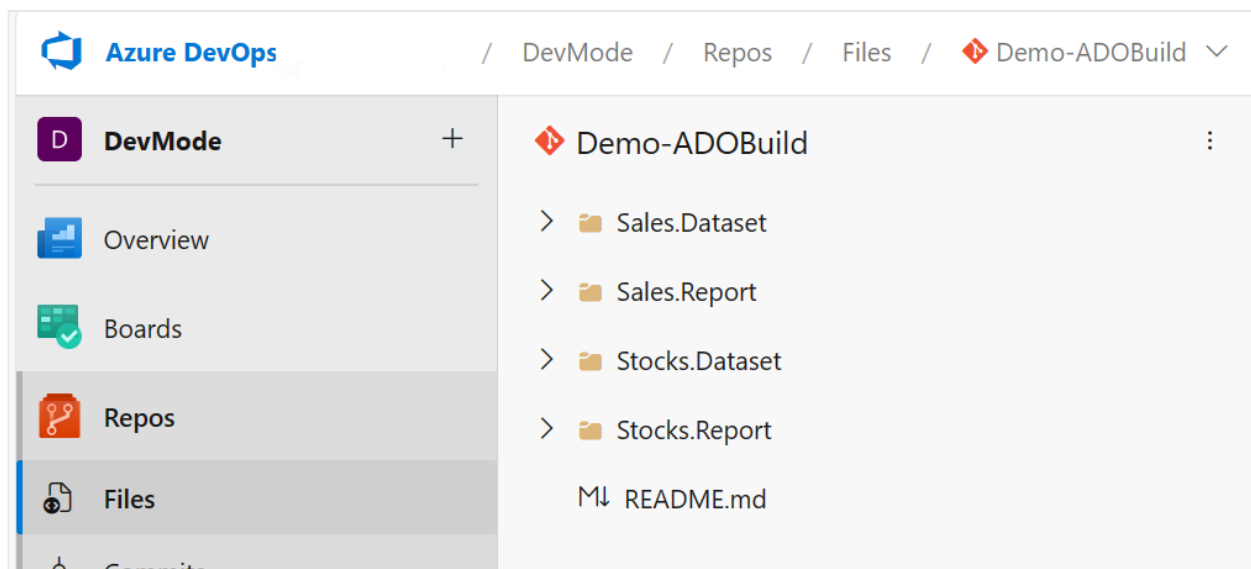
An approach similar to the example in this article would apply to other community tools. This article doesn't delve into the specifics of the community tools mentioned previously nor rule creation and editing. For in-depth information on these topics, refer to the links provided. The focus of this article is on the *process* of establishing a quality gate between source control and Fabric Workspace. It's important to note that the referred community tools are developed by third-party contributors, and Microsoft does not offer support or documentation for them.

Step 1 - Connect Fabric workspace to Azure DevOps

[Connect your Fabric workspace to Azure DevOps:](#)



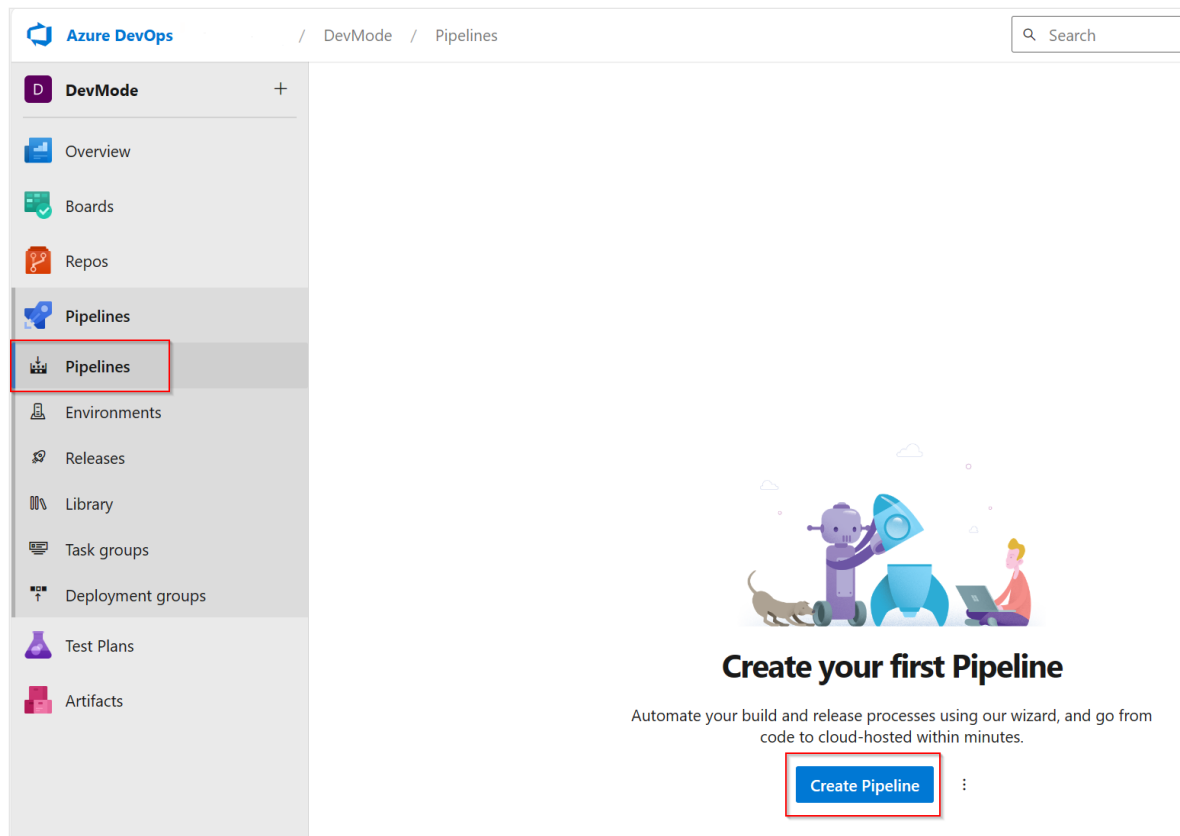
When Fabric Git integration finishes exporting your workspace items, your Azure DevOps branch will contain a folder for each item in your workspace:



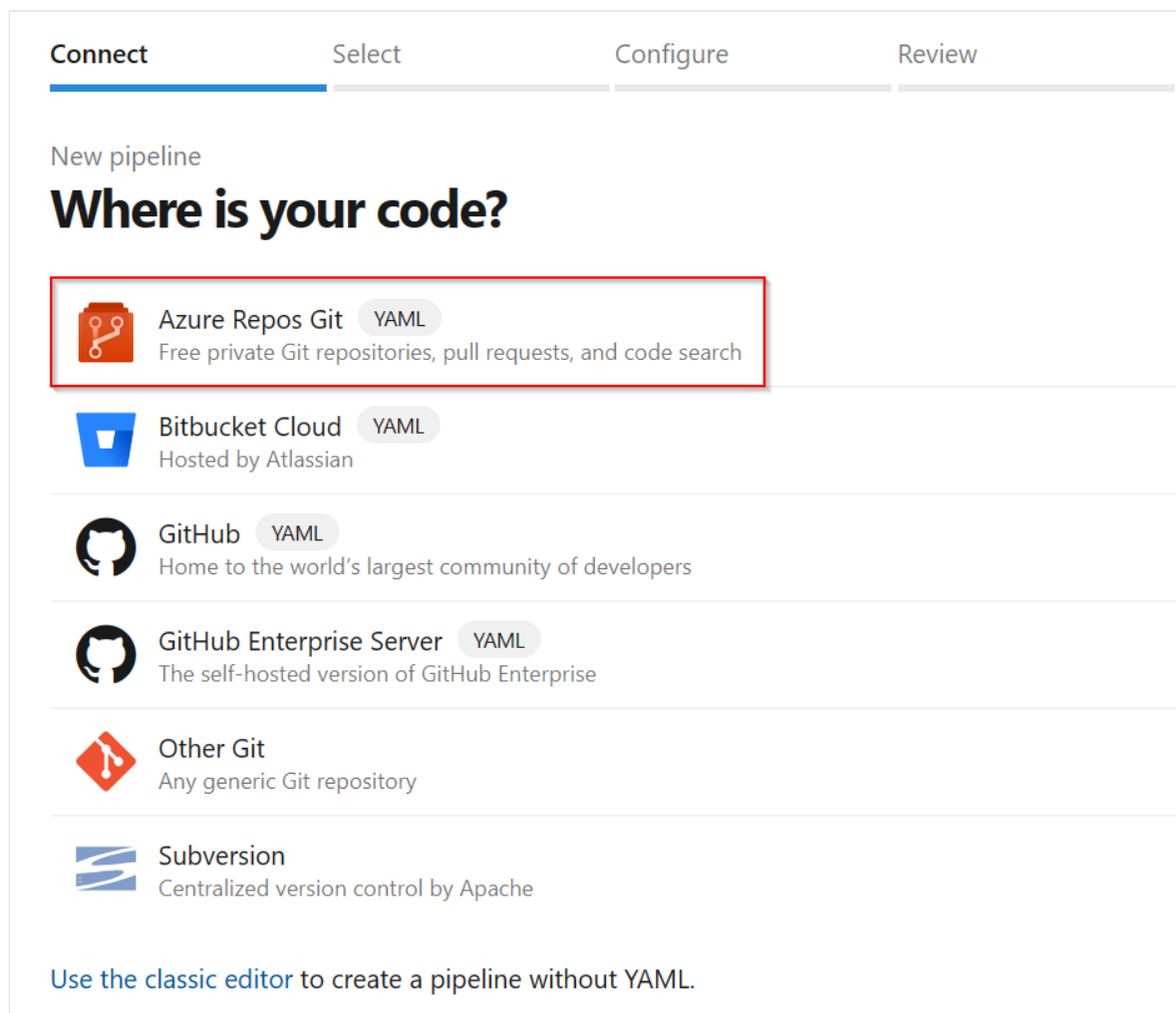
Step 2 - Create and run an Azure DevOps build pipeline

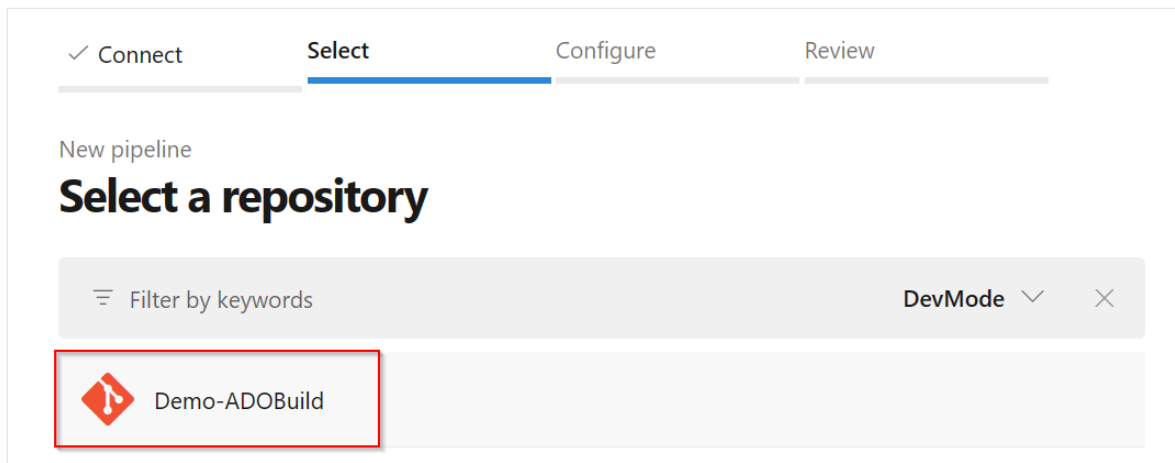
To create a new build pipeline:

1. From the **Pipelines** tab of the left navigation menu, select **Create Pipeline** :

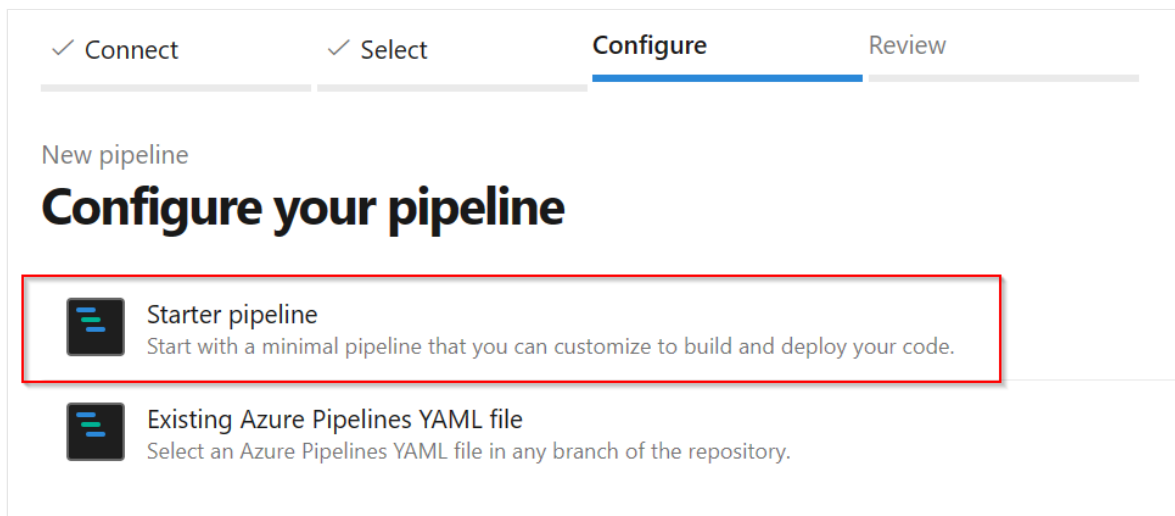


2. Select **Azure Repos Git** and select the first repository (the same repository that's connected to the Fabric workspace):





3. Select Starter pipeline.



The following YAML code appears in the editor:

✓ Connect

✓ Select

✓ Configure

Review

New pipeline

Review your pipeline YAML

◆ Demo-ADOBUILD / azure-pipelines.yml *

1

Starter pipeline

2

Start with a minimal pipeline that you can customize to build and deploy your code.

3

Add steps that build, run tests, deploy, and more:

4

<https://aka.ms/yaml>

5

6

trigger:

7

- main

8

9

pool:

10

- vmImage: ubuntu-latest

11

12

steps:

13

- script: echo Hello, world!

14

- displayName: 'Run a one-line script'

15

16

- script: |

17

echo Add other tasks to build, test, and deploy your project.

18

echo See <https://aka.ms/yaml>

19

- displayName: 'Run a multi-line script'

20

4. Copy and paste the [YAML code from the Power BI developer mode pipeline](#) into the pipeline you created:


Code


Blame


110 lines (91 loc) · 4.53 KB


Code 55% faster with GitHub Copilot


Raw











1

trigger: none

2

3

pool:

4

vmimage: 'windows-latest'

5

6

stages:

7

- stage: Build

8

jobs:

9

10

- job: Build_Datasets

11

steps:

12

- checkout: self

← Demo-ADOBuild

main Demo-ADOBuild / azure-pipelines.yml

```
1 trigger: none
2
3 pool:
4   vmimage: 'windows-latest'
5
6 stages:
7   - stage: Build
8     jobs:
9
10      - job: Build_Datasets
11        steps:
12          - checkout: self
13            path: 'self'
14
15          Settings
16            - task: PowerShell@2
17              displayName: 'Download Tabular Editor and Default Rules'
18              inputs:
19                targetType: inline
20                script: |
21                  $path = "$(Build.SourcesDirectory)"
22                  $tempPath = "$path\_temp"
23                  $toolPath = "$path\_tools\TE"
24                  New-Item -ItemType Directory -Path $tempPath -ErrorAction SilentlyContinue | Out-Null
25                  Write-Host "Downloading Tabular Editor binaries"
26                  $downloadUrl = "https://github.com/otykier/TabularEditor/releases/download/2.18.2/TabularEditor.Portable.zip"
27                  $zipFile = "$tempPath\TabularEditor.zip"
28                  Invoke-WebRequest -Uri $downloadUrl -OutFile $zipFile
29                  Expand-Archive -Path $zipFile -DestinationPath $toolPath -Force
30                  Write-Host "Downloading Dataset default rules"
31                  $downloadUrl = "https://raw.githubusercontent.com/microsoft/Analysis-Services/master/BestPracticeRules/BPARules.json"
32                  Invoke-WebRequest -Uri $downloadUrl -OutFile "$tempPath\Rules-Dataset.json"
```

5. Select Save and Run to commit your new pipeline to the repository.

✓ Connect

✓ Select

✓ Configure

Review

New pipeline

Review your pipeline YAML

Variables

Save and run

Demo-ADOBuild / azure-pipelines.yml * Show assistant

```
1 trigger: none
2
3 pool:
4   vmimage: 'windows-latest'
5
6 stages:
7   - stage: Build
8     jobs:
9
10      - job: Build_Datasets
11        steps:
12          - checkout: self
13            path: 'self'
14
15          Settings
16            - task: PowerShell@2
17              displayName: 'Download Tabular Editor and Default Rules'
```

Save and run



Saving will commit azure-pipelines.yml to the repository.

Commit message

Set up CI with Azure Pipelines

Optional extended description

Add an optional description...

- ☒ Commit directly to the main branch
- ☐ Create a new branch for this commit

Save and run


Azure DevOps runs the pipeline and starts two build jobs in parallel:



#20231003.1 • Set up CI with Azure Pipelines

Demo-ADOBuild

Summary

Triggered by 

Repository and version

📁 Demo-ADOBuild

🌿 main 🔑 1dd4ee5f

Time started and elapsed

📅 Just now

🕒 6s

Jobs

Name




Build_Datasets




Build_Reports

- Build_Datasets
 - Downloads Tabular Editor binaries.
 - Download Best Practice Analyzer [default rules](#). To customize the rules, add a *Rules-Dataset.json* to the root of the repository.
 - Cycle through all the semantic model item folders and run Tabular Editor BPA Rules.
- Build_Reports
 - Download PBI Inspector binaries.
 - Download PBI Inspector [default rules](#). To customize the rules, add a *Rules-Report.json* to the root of the repository.
 - Cycle through all the report item folders and run Power BI Inspector rules.

When it finishes, Azure DevOps creates a report of all the warnings and errors it encountered:


Triggered by 

Repository and version

 Demo-ADOBUILD











 main  1dd4ee5f

Time started and elapsed

 Today at 16:29


 54s

Warnings 42

-  Partition (M - Import) Sales-ddb4c40b-46fd-49ea-9a19-16e7e640a21a violates rule "[Performance] Minimize Power Query transformations"
Build_Datasets • Run Dataset Rules
-  Partition (M - Import) About-77c21240-7751-4575-bf40-8c068bfd01cd violates rule "[Performance] Minimize Power Query transformations"
Build_Datasets • Run Dataset Rules
-  Measure [Sales Amount (% ? LY)] violates rule "[Formatting] Percentages should be formatted with thousands separators and 1 decimal"
Build_Datasets • Run Dataset Rules
-  Measure [Margin %] violates rule "[Formatting] Percentages should be formatted with thousands separators and 1 decimal"
Build_Datasets • Run Dataset Rules
-  Measure [Margin % Overall] violates rule "[Formatting] Percentages should be formatted with thousands separators and 1 decimal"
Build_Datasets • Run Dataset Rules
-  Measure [Value % (? ly)] violates rule "[Formatting] Percentages should be formatted with thousands separators and 1 decimal"
Build_Datasets • Run Dataset Rules
-  Measure [# Customers (with Sales)] violates rule "[Formatting] Whole numbers should be formatted with thousands separators and no decimals"
Build_Datasets • Run Dataset Rules
-  Measure [# Products (with Sales)] violates rule "[Formatting] Whole numbers should be formatted with thousands separators and no decimals"
Build_Datasets • Run Dataset Rules
-  Measure [Sales Qty] violates rule "[Formatting] Whole numbers should be formatted with thousands separators and no decimals"
Build_Datasets • Run Dataset Rules
-  Measure [# Sales] violates rule "[Formatting] Whole numbers should be formatted with thousands separators and no decimals"
Build_Datasets • Run Dataset Rules

[View the log to see the remaining 24 warnings for this task](#)

Select on the link to open a more detailed view of the two build jobs:

-  Measure [# Sales] violates rule "[Formatting] Whole numbers should be formatted with thousands separators and no decimals"
Build_Datasets • Run Dataset Rules

[View the log to see the remaining 24 warnings for this task](#)

← Jobs in run #20231003.1

Demo-ADOBUILD

Build

Build_Datasets19s

Initialize job1s

Checkout Demo-ADOBUILD@main...5s

Download Tabular Editor and Defa...3s

Run Dataset Rules8s

Post-job: Checkout Demo-ADOB...<1s

Finalize Job<1s

Build_Reports15s

Initialize job<1s

Checkout Demo-ADOBUILD@main...4s

Download PBIXInspector3s

Run Report Rules5s

Post-job: Checkout Demo-ADOB...<1s

Finalize Job<1s

Finalize build

Report build status<1s

Run Report Rules

1 Starting: Run Report Rules

2 =====

3 Task : PowerShell

4 Description : Run a PowerShell script on Linux, macOS, or Windows

5 Version : 2.228.1

6 Author : Microsoft Corporation

7 Help : <https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/powershell>

8 =====

9 Generating script.

10 ===== Starting Command Output =====

11 "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoLogo -NoProfile -NonInteractive -ExecutionPolic

12 Running default downloaded rules

13 Processing 'D:\a\1\self\Sales.Report'

14 ##[warning]"Sales" - Rule "Reduce the number of visible visuals on the page" FAILED with result: false, expect

15 ##[warning]"Dynamic Measure" - Rule "Reduce the number of visible visuals on the page" FAILED with result: fal

16 ##[warning]"Sales" - Rule "Reduce the number of objects within visuals" FAILED with result: [

17 "484fbdd73143c5bf71fa",

18 "5ac1caf298449a8acb4"

19], expected: [].

20 ##[warning]"Sales Geo" - Rule "Reduce the number of objects within visuals" FAILED with result: [

21 "a64038428c095bd71739"

22], expected: [].

23 Processing 'D:\a\1\self\Stocks.Report'

24 ##[warning]"Sales" - Rule "Reduce the number of visible visuals on the page" FAILED with result: false, expect

25 ##[warning]"Dynamic Measure" - Rule "Reduce the number of visible visuals on the page" FAILED with result: fal

26 ##[warning]"Sales" - Rule "Reduce the number of objects within visuals" FAILED with result: [

27 "484fbdd73143c5bf71fa",

28 "5ac1caf298449a8acb4"

29], expected: [].

30 ##[warning]"Sales Geo" - Rule "Reduce the number of objects within visuals" FAILED with result: [

31 "a64038428c095bd71739"

32], expected: [].

33 Finishing: Run Report Rules

If your report or semantic model fails a rule with a higher severity level, the build fails, and the error is highlighted:

Manually run by

RR

Repository and version

Demo-ADOBUILD

main 5ca6528b

Errors1

Warnings42

Measure [Margin %] violates rule "[DAX Expressions] Measure references should be unqualified"

Build_Datasets • Run Dataset Rules

Jobs

Name

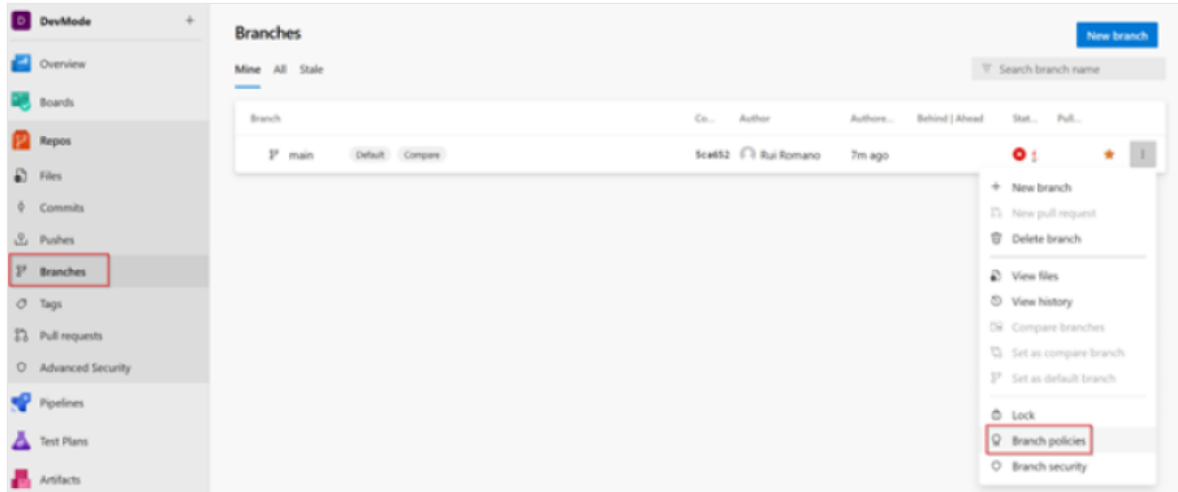
Build_Datasets

Build_Reports

Step 3 - Define branch policies

Once the pipeline is up and running, enable **Branch Policies** on the *main* branch. This step ensures that no commits can be made directly into *main*. A "[pull request](#)" is always required to merge changes back into *main* and you can configure the build pipeline to run with every pull request.

1. Select **Branches** > **main Branch** > **Branch policies**:



2. Configure the created pipeline as a *Build Policy* for the branch:

Add build policy

Build pipeline *
Demo-ADOBuild

Path filter (optional)

Trigger
☒ Automatic (whenever the source branch is updated)
☐ Manual

Policy requirement
☒ Required
 Build must succeed in order to complete pull requests.
☐ Optional
 Build failure will not block completion of pull requests.

Build expiration
☐ Immediately when main is updated
☒ After 12 hours if main has been updated
☐ Never

Display name

Save Cancel

Demo-ADOBuild

Filter by keywords

main Default Compare

Settings Policies Security Approvals and checks

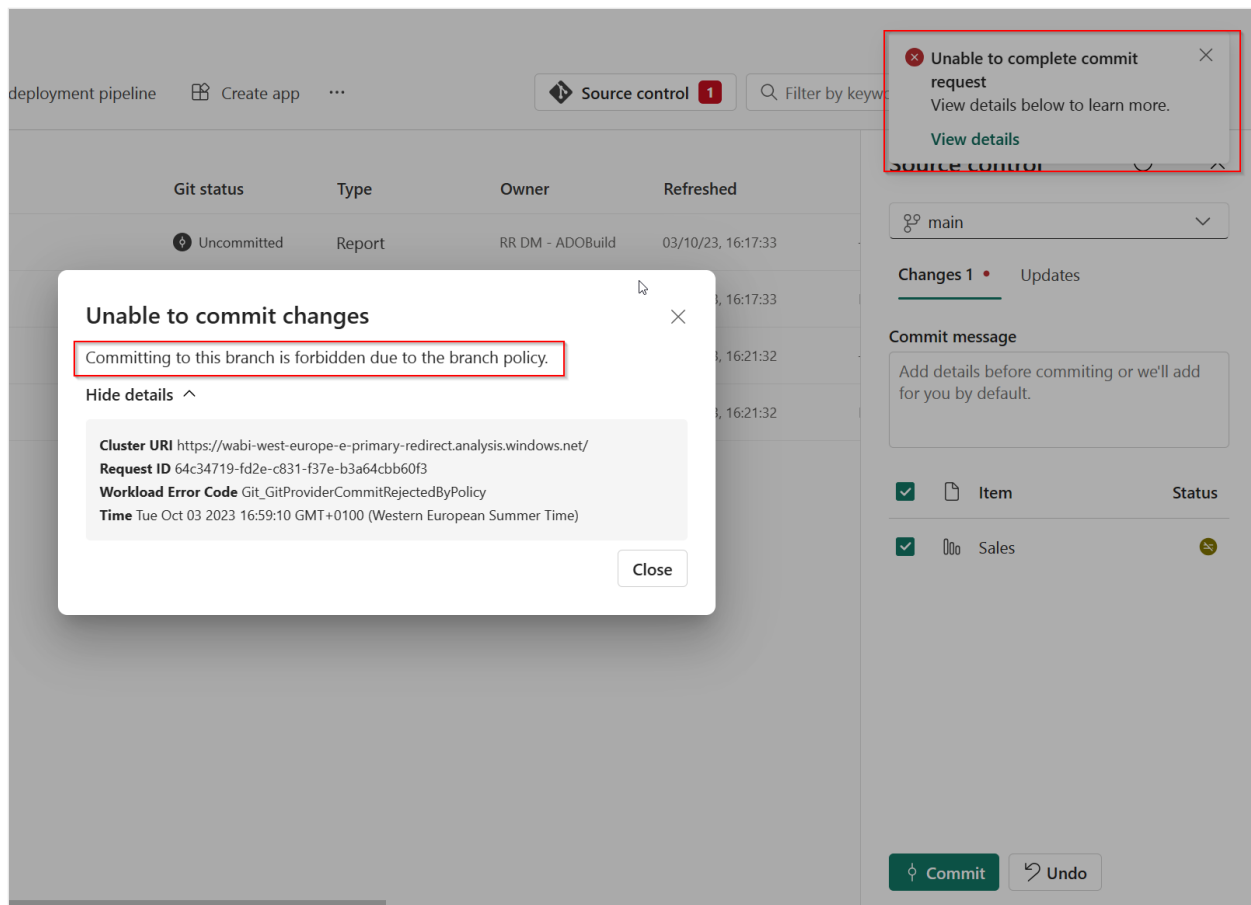
Limit merge types
Off
Control branch history by limiting the available types of merge when pull requests are completed.

Build Validation 1
Validate code by pre-merging and building pull request changes.

| Enabled | Name ↑ | Path filter | Trigger | Inheritance |
|--|---------------------------|-------------|-------------------------------------|-------------|
| <input checked="" type="checkbox"/> On | Demo-ADOBuild Required | | Automatic Expires after 12 hours | |

Step 4 - Create pull request

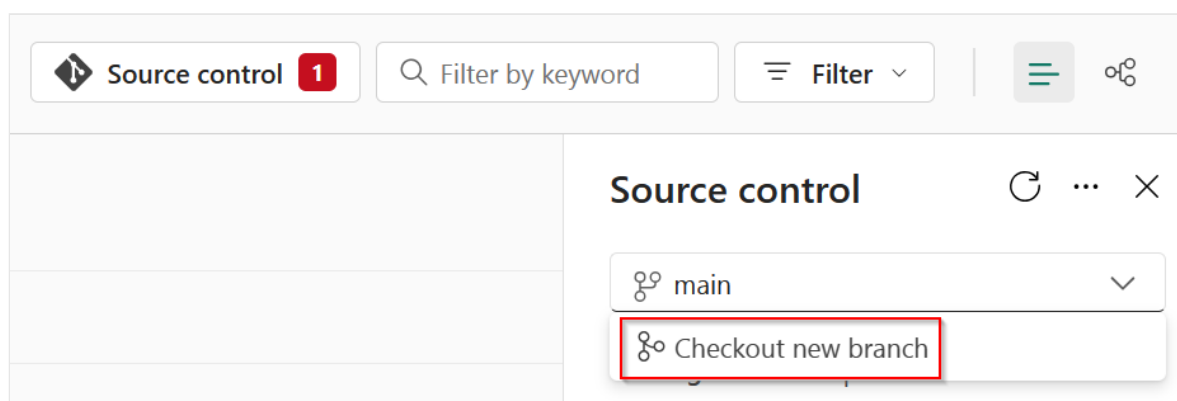
If you return to your Fabric Workspace, make a modification to one of the reports or semantic models, and attempt to commit the change, you receive the following error:

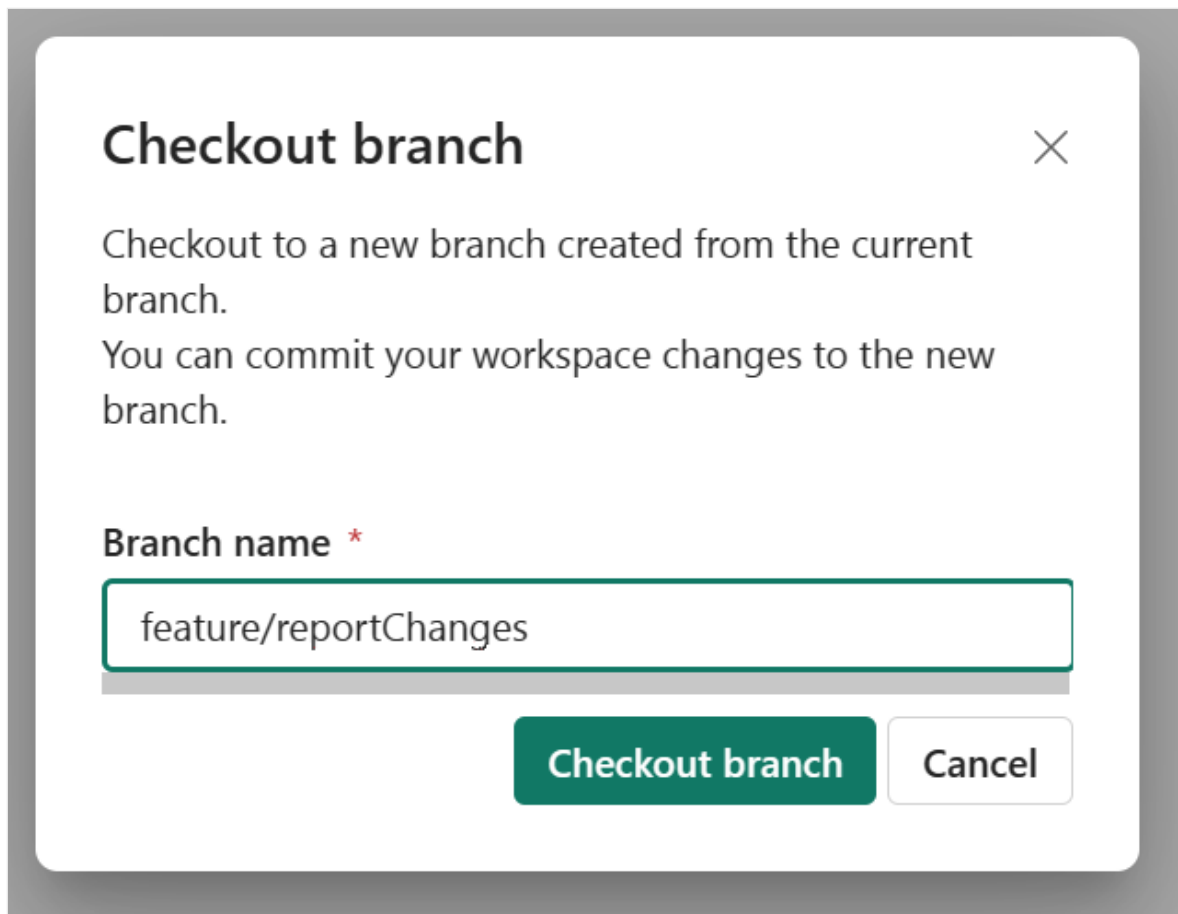


You can only make changes to the main branch through a pull request. To create a pull request checkout a new branch to make the changes on:

Create a branch directly from the Fabric Workspace:

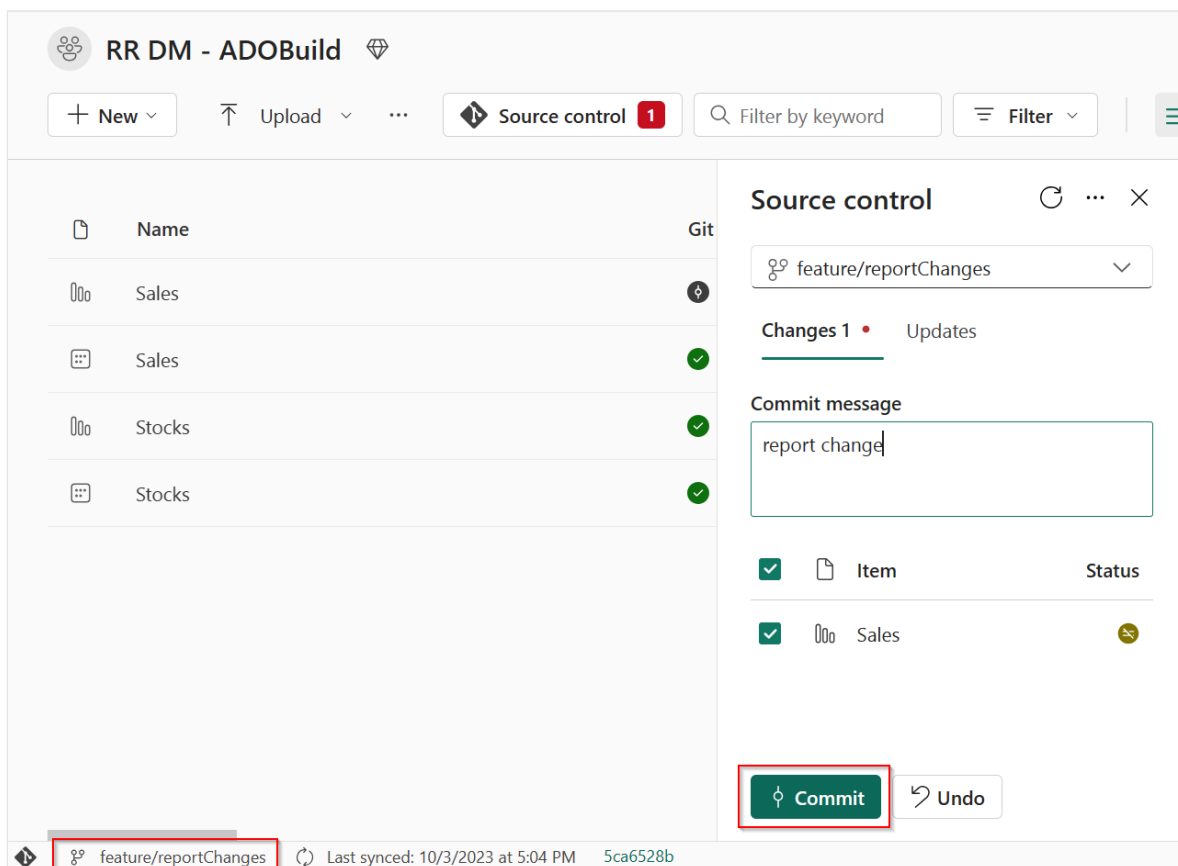
1. In the Source Control pane, select on **Checkout new branch** and provide a name for the branch.



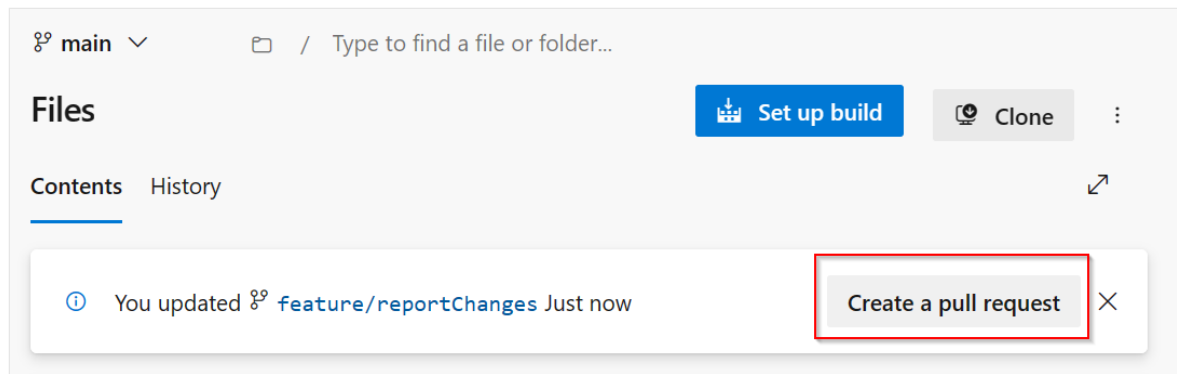


Alternatively, you can choose to develop within a separate, isolated workspace or in Power BI Desktop. For more information, see [develop using another workspace](#)

2. Commit your changes to this new branch.



- Following the commit, create a pull request into the *main* branch from the Azure DevOps portal.



This screenshot shows the 'New pull request' form. At the top, the source branch 'feature/reportChanges' and the target branch 'main' are selected, with a red box around them. Below this are tabs for 'Overview', 'Files', and 'Commits'. The 'Overview' tab is active, showing a 'Title' field with 'report change' and a 'Description' field with 'report change'. Below the description is a rich text editor with various formatting options. At the bottom right, a red box highlights the 'Create' button.

The pull request workflow not only allows you to validate and review the changes, but also automatically triggers the build pipeline.

report change

Active

!14



proposes to merge [feature/reportChanges](#) into [main](#)

Overview

Files

Updates

Commits

Conflicts



1 required check running now

1 optional check not yet run



Demo-ADOBuild Build in progress

[View 2 checks](#)



No merge conflicts

Last checked Just now

Description

report change

If there's a high-severity build error in one of the rules, you can't finalize the pull request and merge the changes back into the main branch.

The screenshot shows a pull request interface for a repository named "Demo-ADOBuild". The pull request is titled "report change" and is proposed by Rui Romano to merge "feature/reportChanges" into "main". The status is "Active" with 114 comments. The "Overview" tab is selected, showing a list of checks. One check, "Demo-ADOBuild Build failed", is highlighted with a red box. Below it, a specific error message is shown: "21 Measure [Margin %] violates rule '[DAX Expressions] Measure references should be unique'". A green checkmark indicates "No merge conflicts" as of 2m ago. The "Description" section contains the text "report change". At the bottom, there is a comment input field and a list of activities, including "Rui Romano created the pull request".

On the right, a "Complete pull request" dialog is open. It has a red box around the "Build failed" status at the top. The "Merge type" is set to "Merge (no fast forward)". Under "Post-completion options", the checkboxes for "Complete associated work items after merging" and "Delete feature/reportChanges after merging" are checked, while "Customize merge commit message" is unchecked. At the bottom of the dialog, there are "Cancel" and "Complete merge" buttons, with the latter highlighted by a red box.

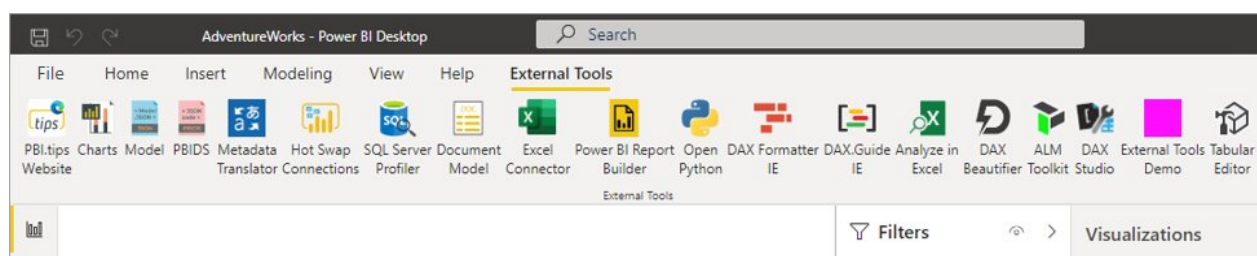
Learn more about PBIP and Fabric Git Integration in [blog post](#).

External tools in Power BI Desktop

Article • 11/10/2023

Power BI has a vibrant community of business intelligence professionals and developers. Community contributors create free tools that use Power BI and Analysis Services APIs to extend and integrate with Power BI Desktop's data modeling and reporting features.

The **External Tools** ribbon provides easy access to external tools that are installed locally and *registered* with Power BI Desktop. When launched from the External Tools ribbon, Power BI Desktop passes the name and port number of its internal data model engine instance and the current model name to the tool. The tool then automatically connects, providing a seamless connection experience.



External tools generally fall into one of the following categories:

Semantic modeling - Open-source tools such as DAX Studio, ALM Toolkit, Tabular Editor, and Metadata Translator extend Power BI Desktop functionality for specific data modeling scenarios such as Data Analysis Expressions (DAX) query and expression optimization, application lifecycle management (ALM), and metadata translation.

Data analysis - Tools for connecting to a model in read-only to query data and perform other analysis tasks. For example, a tool might launch Python, Excel, and Power BI Report Builder. The tool connects the client application to the model in Power BI Desktop for testing and analysis without having to first publish the Power BI Desktop (*.pbix*) file to the Power BI service. Tools to document a Power BI semantic model also fall into this category.

Miscellaneous - Some external tools don't connect to a model at all, but instead extend Power BI Desktop to make helpful tips and make helpful content more readily accessible. For example, PBI.tips tutorials, DAX Guide from sqlbi.com, and the PowerBI.tips Product Business Ops community tool, make installation of a large selection of external tools easier. These tools also assist registration with Power BI Desktop, including DAX Studio, ALM Toolkit, Tabular Editor, and many others easy.

Custom - Integrate your own scripts and tools by adding a *.pbtool.json document to the Power BI Desktop\External Tools folder.

Before installing external tools, keep the following notes in mind:

- External tools aren't supported in Power BI Desktop for Power BI Report Server.
- External tools are provided by external, third-party contributors. Microsoft doesn't provide support or documentation for external tools.

Featured open-source tools

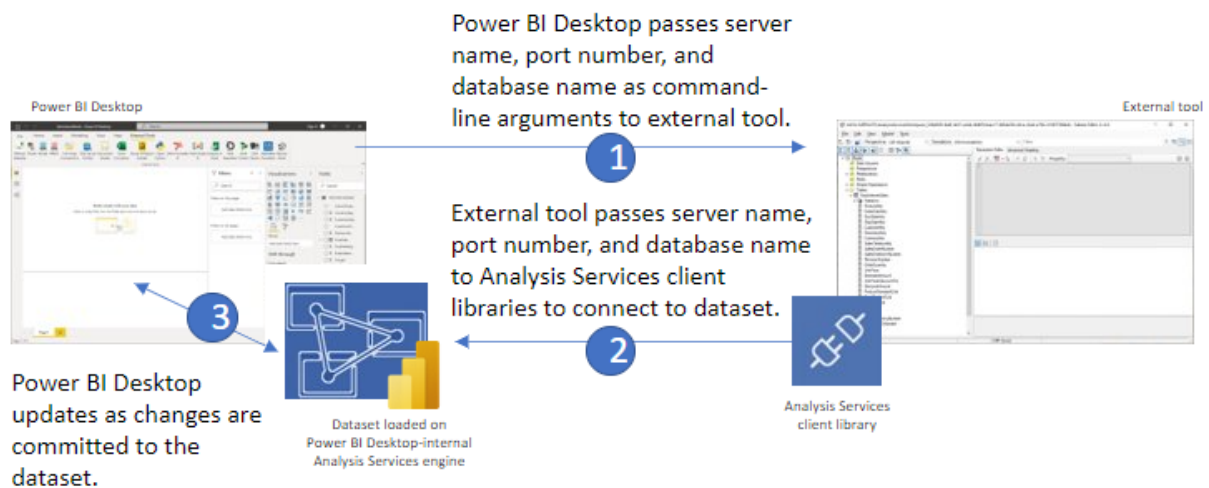
There are many external tools out there. Here are some of the most popular and belong in every Power BI Desktop data modelers toolbox:

| Tool | Description |
|-----------------------------|---|
| PowerBI.tips - Business Ops | An easy to use deployment tool for adding external tools extensions to Power BI Desktop. The Business Ops goal is to provide a one stop shop for installing all the latest versions of external tools. To learn more, go to PowerBI.tips - Business Ops . |
| Tabular Editor | Model creators can easily build, maintain, and manage tabular models by using an intuitive and lightweight editor. A hierarchical view shows all objects in your tabular model organized by display folders, with support for multi-select property editing and DAX syntax highlighting. To learn more, go to tabulareditor.com . |
| DAX Studio | A feature-rich tool for DAX authoring, diagnosis, performance tuning, and analysis. Features include object browsing, integrated tracing, query execution breakdowns with detailed statistics, DAX syntax highlighting and formatting. To get the latest, go to DAX Studio on GitHub. |
| ALM Toolkit | A schema compare tool for Power BI models and semantic models, used for application lifecycle management (ALM) scenarios. You can perform straightforward deployment across environments and retain incremental refresh historical data. You can diff and merge metadata files, branches, and repos. You can also reuse common definitions between semantic models. To get the latest, go to alm-toolkit.com . |
| Metadata Translator | Streamlines localization of Power BI models and semantic models. The tool can automatically translate captions, descriptions, and display folder names of tables, columns, measures, and hierarchies. The tool translates by using the machine translation technology of Azure Cognitive Services. You can also export and import translations via Comma Separated Values (.csv) files for convenient bulk editing in Excel or a localization tool. To get the latest, go to Metadata Translator on GitHub. |

External tools integration architecture

Power BI Desktop (*pbix*) files consist of multiple components including the report canvas, visuals, model metadata, and any data that was loaded from data sources. When Power BI Desktop opens a *pbix* file, it launches an Analysis Services process in the background to load the model so that the data modeling features and report visuals can access model metadata and query model data.

When Power BI Desktop launches Analysis Services as its analytical data engine, it dynamically assigns a random port number. It also loads the model with a randomly generated name in the form of a globally unique identifier (GUID). Because these connection parameters change with every Power BI Desktop session, it's difficult for external tools to discover on their own the correct Analysis Services instance and model to connect to. External tools integration solves this problem by allowing Power BI Desktop to send the Analysis Services server name, port number, and model name to the tool as command-line parameters when starting the external tool from the External Tools ribbon, as shown in the following diagram.



With the Analysis Services Server name, port number, and model name, the tool uses Analysis Services client libraries to establish a connection to the model, retrieve metadata, and execute DAX or MDX queries. Whenever an external data modeling tool updates the metadata, Power BI Desktop synchronizes the changes so that the Power BI Desktop user interface reflects the current state of the model accurately. Keep in mind there are some limitations to the synchronization capabilities as described later.

Data modeling operations

External tools, which connect to Power BI Desktop's Analysis Services instance, can make changes (write operations) to the data model. Power BI Desktop then synchronizes those changes with the report canvas so they're shown in report visuals. For example, external data modeling tools can override the original format string expression of a

measure, and edit any of the measure properties including KPIs and detail rows. External tools can also create new roles for object and row-level security, and add translations.

Supported write operations

Objects that support write operations:

| Object | Connect to AS instance |
|-----------------------------|------------------------|
| Tables | No |
| Columns | Yes ¹ |
| Calculated tables | Yes |
| Calculated columns | Yes |
| Relationships | Yes |
| Measures | Yes |
| Model KPIs | Yes |
| Calculation groups | Yes |
| Perspectives | Yes |
| Translations | Yes |
| Row Level Security (RLS) | Yes |
| Object Level Security (OLS) | Yes |
| Annotations | Yes |
| M expressions | No |

¹ - When using external tools to connect to the AS instance, changing a column's data type is supported, however, renaming columns is not supported.

Power BI Desktop *project files* offer a broader scope of supported write operations. Those objects and operations that don't support write operations by using external tools to connect to Power BI Desktop's Analysis Services instance may be supported by editing Power BI Desktop project files. To learn more, see [Power BI Desktop projects - Model authoring](#).

Data modeling limitations

All Tabular Object Model (TOM) metadata can be accessed for read-only. Write operations are limited because Power BI Desktop must remain in-sync with the external modifications, therefore the following operations aren't supported:

- Any TOM object types not covered in Supported write operations, such as tables and columns.
- Editing a Power BI Desktop template (PBIT) file.
- Report-level or data-level translations.
- Renaming tables and columns isn't yet supported
- Sending processing commands to a semantic model loaded in Power BI Desktop

Registering external tools

External tools are *registered* with Power BI Desktop when the tool includes a *.pbtool.json registration file in the `C:\Program Files (x86)\Common Files\Microsoft Shared\Power BI Desktop\External Tools` folder. When a tool is registered, and includes an icon, the tool appears in the External Tools ribbon. Some tools, like ALM Toolkit and DAX Studio create the registration file automatically when you install the tool. However, many tools, like SQL Profiler typically don't because the installer they do have doesn't include creating a registration file for Power BI Desktop. Tools that don't automatically register with Power BI Desktop can be registered manually by creating a *.pbtool.json registration file.

To learn more, including json examples, see [Register an external tool](#).

Disabling the External Tools ribbon

The External Tools ribbon is enabled by default, but can be disabled by using Group Policy or editing the **EnableExternalTools** registry key directly.

- Registry key: `Software\Policies\Microsoft\Power BI Desktop\`
- Registry value: `EnableExternalTools`

A value of 1 (decimal) enables the External Tools ribbon, which is also the default value.

A value of 0 (decimal) disable the External Tools ribbon.

See also

[Register an external tool](#)

Register an external tool

Article • 11/10/2023

Some tools must be manually registered with Power BI Desktop. To register an external tool, create a JSON file with the following example code:

JSON

```
{
  "name": "<tool name>",
  "description": "<tool description>",
  "path": "<tool executable path>",
  "arguments": "<optional command line arguments>",
  "iconData": "image/png;base64,<encoded png icon data>"
}
```

The pbtool.json file includes the following elements:

- **name:** Provide a name for the tool, which will appear as a button caption in the External Tools ribbon within Power BI Desktop.
- **description:** (optional) Provide a description, which will appear as a tooltip on the External Tools ribbon button within Power BI Desktop.
- **path:** Provide the fully qualified path to the tool executable.
- **arguments:** (optional) Provide a string of command-line arguments that the tool executable should be launched with. You can use any of the following placeholders:
 - **%server%:** Replaced with the server name and portnumber of the local instance of Analysis Services Tabular for imported/DirectQuery data models.
 - **%database%:** Replaced with the database name of the model hosted in the local instance of Analysis Services Tabular for imported/DirectQuery data models.
- **iconData:** Provide image data, which will be rendered as a button icon in the External Tools ribbon within Power BI Desktop. The string should be formatted according to the syntax for Data URIs without the "data:" prefix.

Name the file "`<tool name>.pbtool.json`" and place it in the following folder:

- `%commonprogramfiles%\Microsoft Shared\Power BI Desktop\External Tools`

For 64-bit environments, place the files in the following folder:

- **Program Files (x86)\Common Files\Microsoft Shared\Power BI Desktop\External Tools**

Files in that specified location with the **.pbtool.json** extension are loaded by Power BI Desktop upon startup.

Example

The following *.pbtool.json file launches powershell.exe from the External Tools ribbon and runs a script called pbiToolsDemo.ps1. The script passes the server name and port number in the -Server parameter and the semantic model name in the -Database parameter.

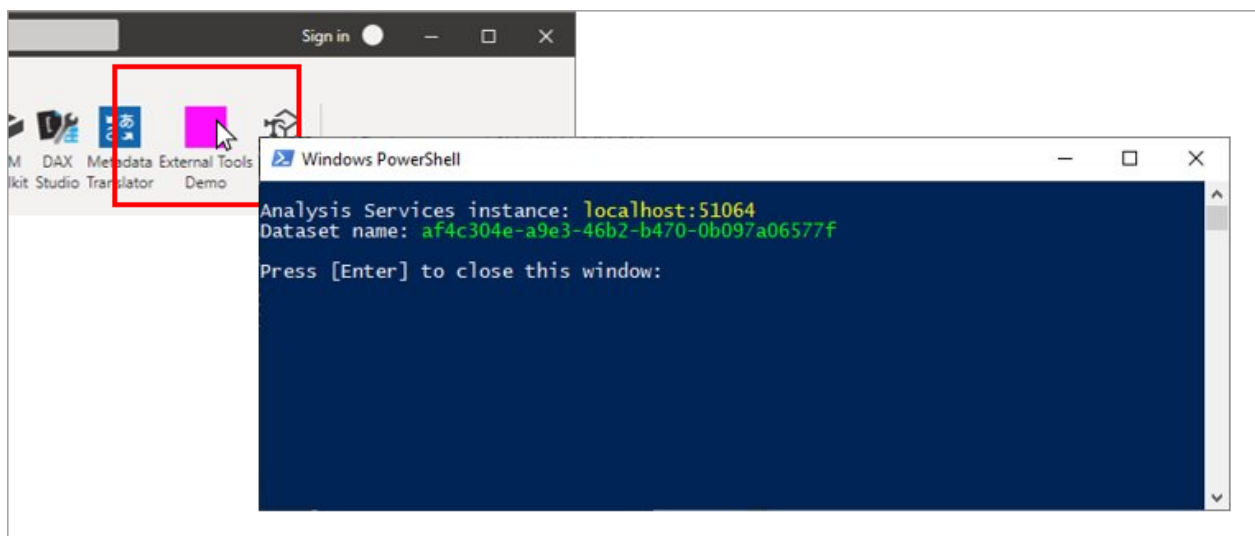
JSON

```
{
  "version": "1.0.0",
  "name": "External Tools Demo",
  "description": "Launches PowerShell and runs a script that outputs server and database parameters. (Requires elevated PowerShell permissions.)",
  "path": "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
  "arguments": "C:\\pbiToolsDemo.ps1 -Server \"%server%\" -Database \"%database%\"",
  "iconData": "image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAyAAAFfcSJAAAAAXNSR0IArs4c6QAAARnQU1BAACxjwv8YQUAAAJcEhZcwAADsEAAA7BAbiRa+0AAAAANSURVBhXY/jH9+8/AAciAwpql7QkAAAAE1FTkSuQmCC"
}
```

The corresponding pbiToolsDemo.ps1 script outputs the Server and Database parameters to the console.

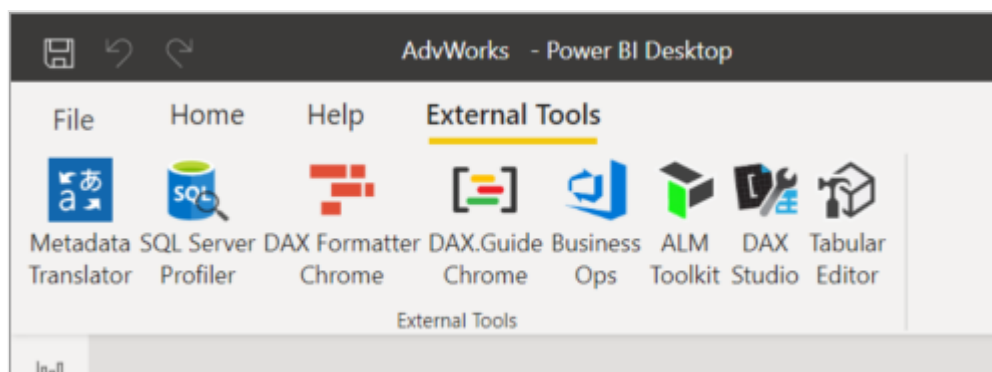
PowerShell

```
[CmdletBinding()]
param
(
    [Parameter(Mandatory = $true)]
    [string] $Server,
    [Parameter(Mandatory = $true)]
    [string] $Database
)
Write-Host ""
Write-Host "Analysis Services instance: " -NoNewline
Write-Host "$Server" -ForegroundColor Yellow
Write-Host "Dataset name: " -NoNewline
Write-Host "$Database" -ForegroundColor Green
Write-Host ""
Read-Host -Prompt 'Press [Enter] to close this window'
```



Icon data URIs

To include an icon in the External Tools ribbon, the pbtool.json registration file must include an iconData element.



The iconData element takes a data URI without the **data:** prefix. For example, the data URI of a one pixel magenta png image is:

```
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAFScJAAAAAXNSR0IArs4c6QAAAArnQU1BAACxjwv8YQUAAAJcEhZcwAADsEAAA7BAbiRa+0AAAANSURBVVBhXY/jH9+8/AACiAwpq17QkAAAAE1FTkSuQmCC
```

Be sure to remove the **data:** prefix, as shown in the pbtool.json preceding example.

To convert a .png or other image file type to a data URI, use an online tool or a custom tool such as the one shown in the following C# code snippet:

c#

```
string ImageDataUri;
OpenFileDialog openFileDialog1 = new OpenFileDialog();
openFileDialog1.Filter = "PNG Files (.png)|*.png|All Files (*.*)|*.*";
openFileDialog1.FilterIndex = 1;
openFileDialog1.Multiselect = false;
```

```
openFileDialog1.CheckFileExists = true;
bool? userClickedOK = openFileDialog1.ShowDialog();
if (userClickedOK == true)
{
    var fileName = openFileDialog1.FileName;
    var sb = new StringBuilder();
    sb.Append("image/")
        .Append((System.IO.Path.GetExtension(fileName) ??
"png").Replace(".", ""))
        .Append(";base64,")
        .Append(Convert.ToBase64String(File.ReadAllBytes(fileName)));
    ImageDataUri = sb.ToString();
}
```

See also

[External tools in Power BI Desktop](#)

[Analysis Services client libraries](#)

[Tabular Object Model \(TOM\)](#)