Power BI developer solutions documentation

Solutions and best practices for Power BI developers of all kinds. 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. Microsoft Power BI Desktop developer mode brings Pro BI developer experiences right into Power BI Desktop.

Power BI Desktop projects

CONCEPT

Power BI Desktop projects overview

Project semantic model folder

Project report folder

Project Git integration

Project Azure DevOps integration

Automation with Power BI REST APIs

CONCEPT

Using the Fabric REST APIs

Using the Power BI REST APIs

Interact with Power BI reports using the Power BI REST API

Power BI CI/CD

CONCEPT

Introduction to deployment pipelines

Azure DevOps Integration

Continuous integration with Azure DevOps and Git

CI/CD with Git and Deployment Pipelines

Solutions and Best Practices

CONCEPT

Hot and cold table partitions to optimize very large Power BI data models

External tools in Power BI Tabular Object Model (TOM)

Programming Power BI semantic models with the Tabular Object Model (TOM)

External tools

CONCEPT

External tools in Power BI Desktop

Register an external tool

Power BI Desktop projects (PREVIEW)

Article • 01/20/2025

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

> **Tip**
>
> For guidance about how to plan a Power BI development, see **Power BI implementation planning**.

Power BI Desktop introduces a new way to author, collaborate, and save your projects. When you save your work as a ***Power BI Project*** (PBIP), report and semantic model *item* 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** - Item definition files are formatted text files containing semantic model and report metadata. These files are 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 item definitions** - You can programmatically generate and modify item definition text files, enabling batch operations such as updating all report pages visuals or adding a set of measures to each table. For semantic models, you can use the Tabular Object Model (TOM) client library to deserialize the semantic model metadata, make programmatic modifications, and serialize it back to the files.

- **Source control** - Power BI semantic model and report item 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.

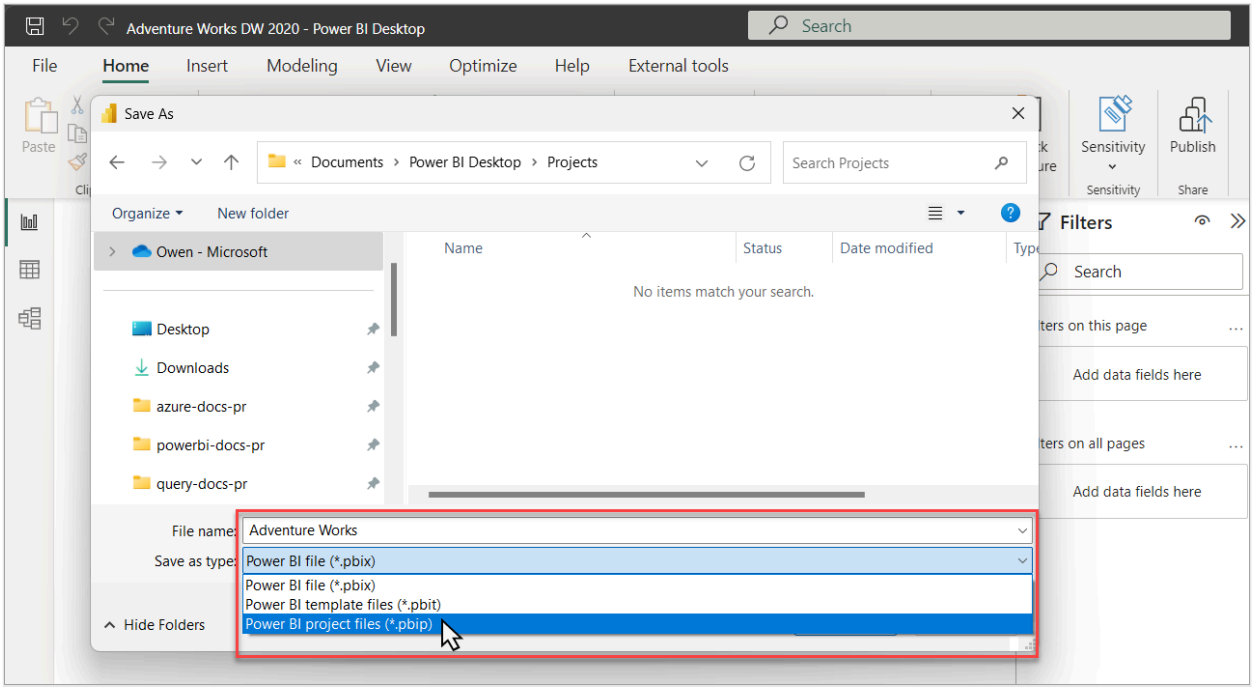
Enable preview features

Saving as a Power BI Project in Power BI Desktop is currently in **preview**, and you must enable it in **Preview features**.

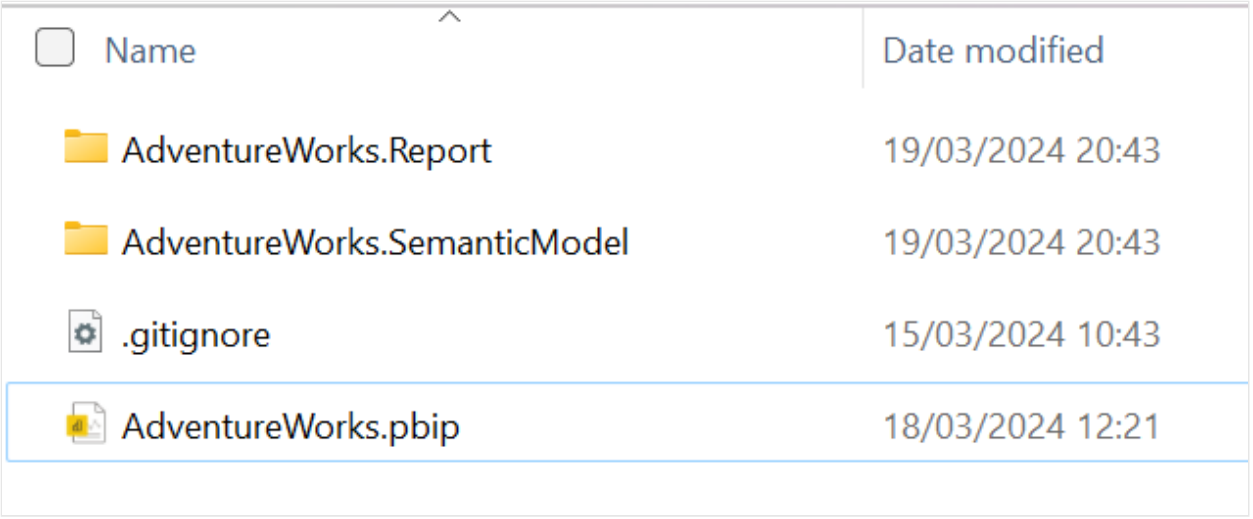
Go to **File** > **Options and settings** > **Options** > **Preview features** and check the box next to **Power BI Project (.pbip) save option**.

Save as a project

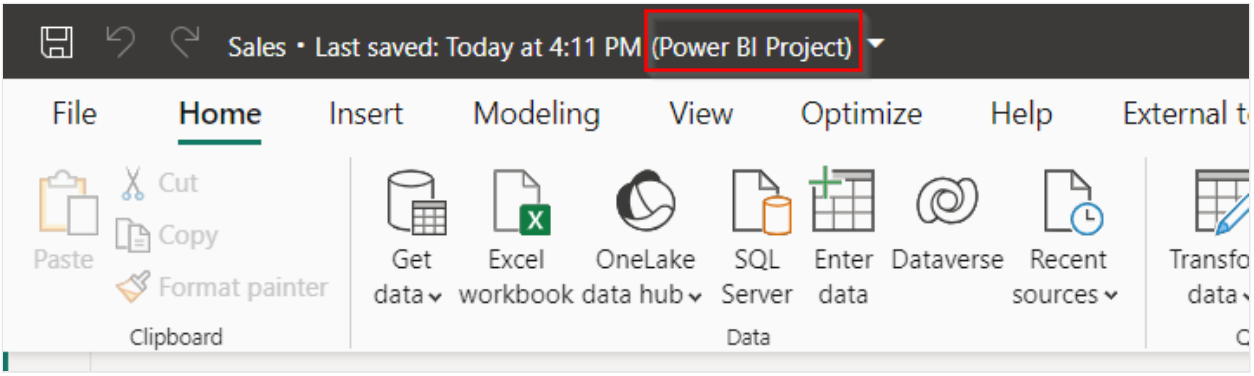
If you're working on a new project or you opened an existing Power BI Desktop file (pbix), you can save your work as a Power BI *project* file (pbip):



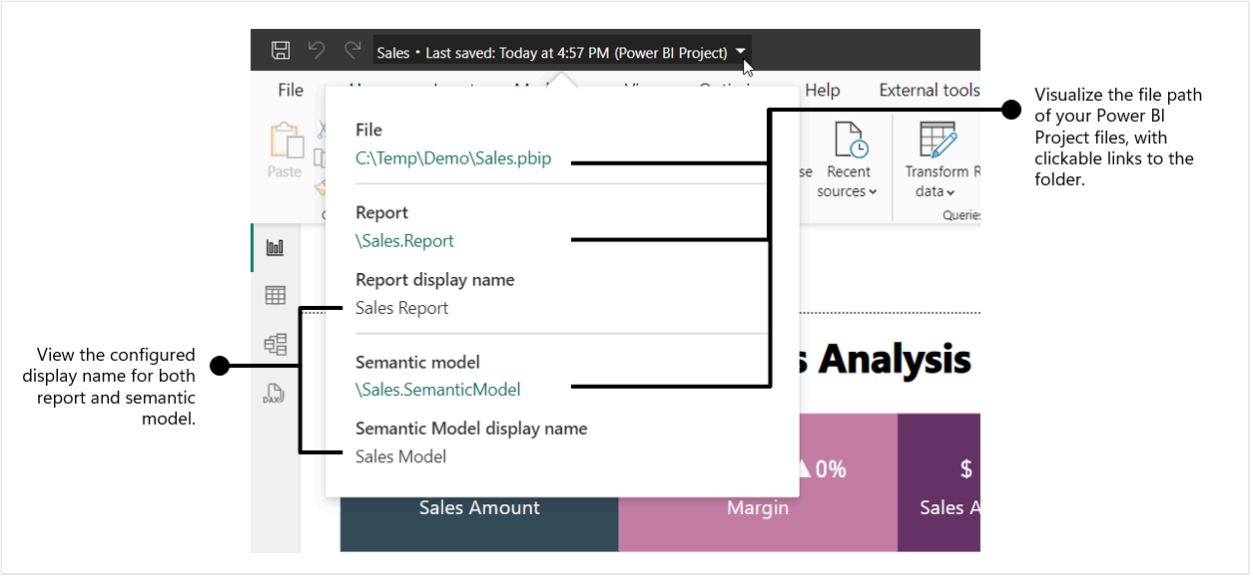
When you save as a project, Power BI Desktop saves report and semantic model items as folders, each containing text files that define the item:



After saving as a project, you can see when you're working on a project by looking at the title bar:



If you select on the title bar, a flyout appears that's specific for Power BI Project. This flyout lets you locate the project files and the display name settings for the report and the semantic model. You can also open the folder in file explorer by clicking on the paths.



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

<project name>.SemanticModel

A collection of files and folders that represent a Power BI semantic model. To learn more about the files and subfolders and files in here, see Project Semantic Model 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 for Power BI Project files, such as the cache.abf and localSettings.json.

Power BI Desktop creates the .gitignore file only if one doesn't already exist in the chosen save folder or parent Git repository.

Default content of .gitignore when saving as PBIP:

```
**/.pbi/localSettings.json
**/.pbi/cache.abf
```

<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, see 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 semantic model, if there's a relative reference to a semantic model.

You can save multiple reports and semantic models 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.

Markdown

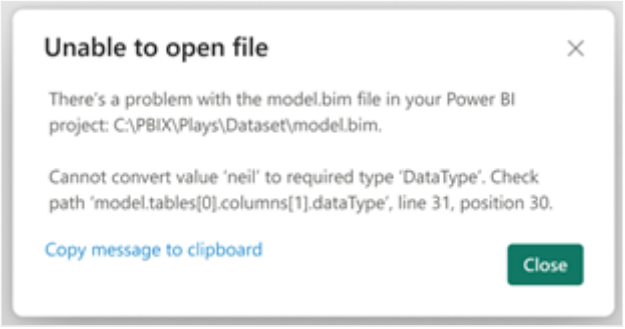
```
├── project
│   ├── AdventureWorks-Sales.Report
│   │   └── definition.pbir
│   ├── AdventureWorks-Stocks.Report
│   │   └── definition.pbir
│   ├── AdventureWorks.SemanticModel
│   │   └── *.*
│   .gitignore
└──  AdventureWorks.pbip
```

Changes outside Power BI Desktop

When saved as a project, you're not forced into making changes to your semantic model 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
  - semanticModelDiagramLayout.json
- SemanticModel\
  - diagramLayout.json

Deploy to Fabric workspace

When working with Power BI project files, you can deploy your content to a Fabric workspace using the following publishing mechanisms:

- Use Fabric Git Integration.
- Use Fabric APIs.
- Use Power BI Desktop publish option.

> **Note**
>
> Publishing through **Power BI Desktop publish** uses a temporary PBIX file that is published to the service, similar to saving and publishing a PBIX file. Unlike other PBIP deployment options that only deploy metadata, this publishing method deploys both the metadata and the **local data cache** of the semantic model being edited.

Model authoring

You can make changes to the semantic 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 those supported can cause unexpected results.

Objects that support write operations:

Expand table

Object	Connect to AS instance	File change / TMDL view
Tables	No	Yes
Columns	Yes [1], [2]	Yes
Calculated tables	Yes	Yes
Calculated columns	Yes	Yes
Hierarchies	Yes	Yes

Object	Connect to AS instance	File change / TMDL view
Relationships	Yes	Yes
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 semantic model 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 might 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 semantic model 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.
- Modifying table query expressions outside of Power BI Desktop results in the removal of the table data upon restarting Power BI Desktop.

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.
- 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 and SharePoint.
- When editing PBIP files outside of Power BI Desktop, they should be saved using UTF-8 without BOM encoding.
- Report Linguistic Schema isn't 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.
- Power BI Projects is currently not supported in Microsoft Power BI Desktop version optimized for Power BI Report Server.

- Live connect reports saved as PBIP require the XMLA Endpoint to be enabled.

Frequently asked questions

Question: Looking at semantic model and report item 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.

Question: Why isn't there a *.pbip file when I connect my Fabric workspace to Git? How can I edit my report and semantic model in Power BI Desktop?

Answer: The PBIP file is optional and simply serves as a shortcut to the report folder. You can open both the report and the semantic model for editing in Power BI Desktop by opening the definition.pbir file located in the report folder.

Related content

- Power BI Desktop project semantic model 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

Feedback

Was this page helpful? Yes No

Provide product feedback | Ask the community

Power BI Desktop project semantic model folder

Article • 08/13/2024

> **ⓘ Important**
>
> Power BI Desktop projects is currently in **preview**.

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

- .pbi\
  - localSettings.json
  - editorSettings.json
  - cache.abf
  - unappliedChanges.json
- definition.pbism[1]
- model.bim[2]
- definition\ folder[3]
- diagramLayout.json
- .platform

1 - This file is required.
2 - This file is required when saving using TMSL format.
3 - This file is required when saving using TMDL format.

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

Semantic Model files

.pbi\localSettings.json

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

For more information, see the localSettings.json schema document.

.pbi\editorSettings.json

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

For more information, see the editorSettings.json schema document.

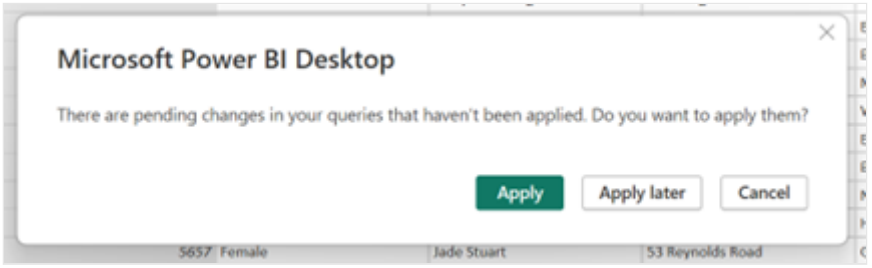
.pbi\cache.abf

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, Git ignores this file.

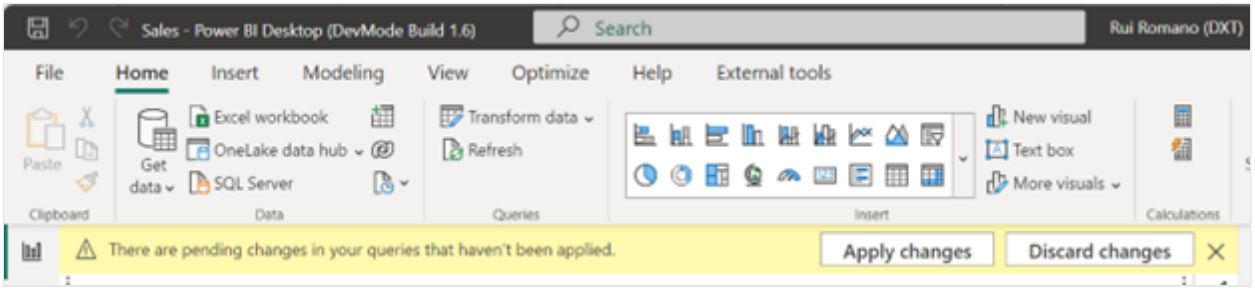
Power BI Desktop can open a project without a cache.abf file. In that case, it opens the report connected to a model with its entire definition but without data. If a cache.abf exists, Power BI Desktop loads the data and overwrites the model definition with the content in model.bim.

.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's 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 semantic model 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 can exclude this file from Git's tracking, preventing unfinished query work from affecting other developers.

For more information, see the unappliedChanges.json schema document.

definition.pbism

Contains the overall definition of a semantic model and core settings.

This file also specifies the supported semantic model definition formats through the 'version' property.

Expand table

Version	Supported formats
1.0	Semantic model definition must be stored as TMSL in the model.bim file.
4.0 or above	Semantic model definition can be stored as TMSL (model.bim file) or TMDL (\definition folder).

For more information, see the definition.pbism schema document.

model.bim

This file is only available if the Power BI project is saved using the TMSL format. It contains a Tabular Model Scripting Language (TMSL) Database object definition of the project model.

definition\ folder

This folder is only available if the Power BI project is saved using the TMDL format. It replaces the model.bim file.

This folder contains a Tabular Model Definition Language (TMDL) Database object definition of the project model.

diagramLayout.json

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

.platform

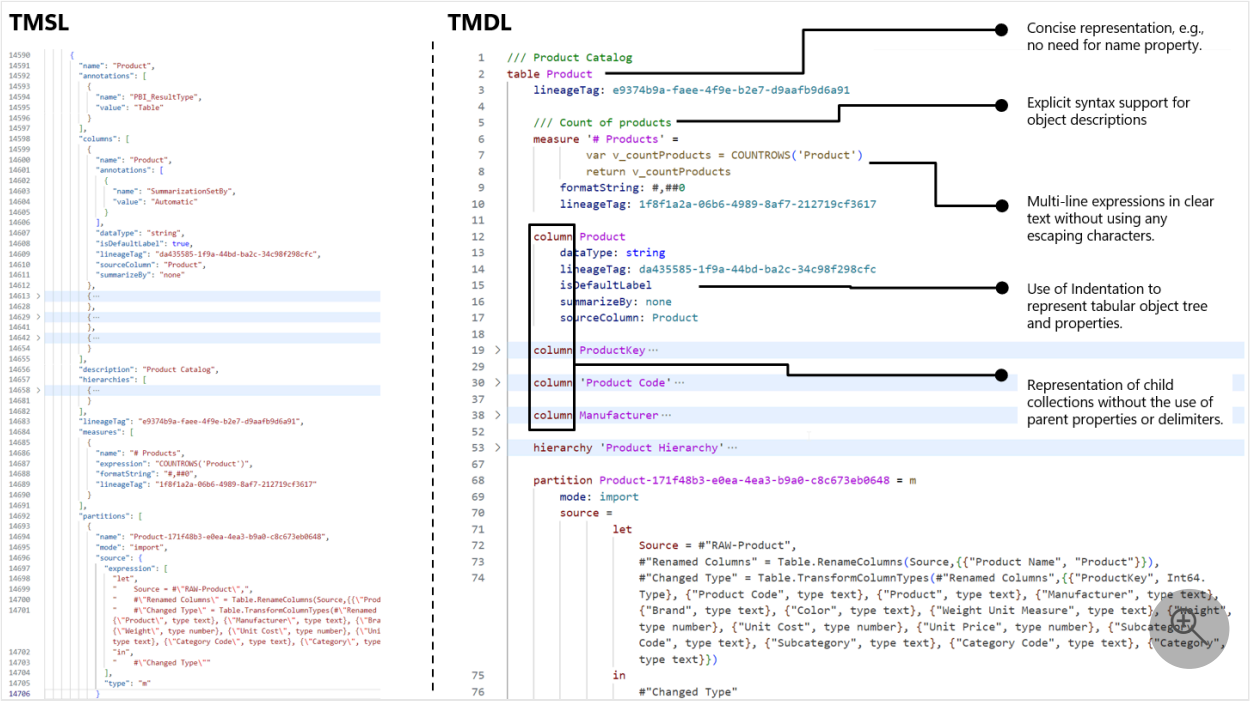
Fabric platform file that holds properties vital for establishing and maintaining the connection between Fabric items and Git.

To learn more, see Git integration automatically generated system files.

TMDL format

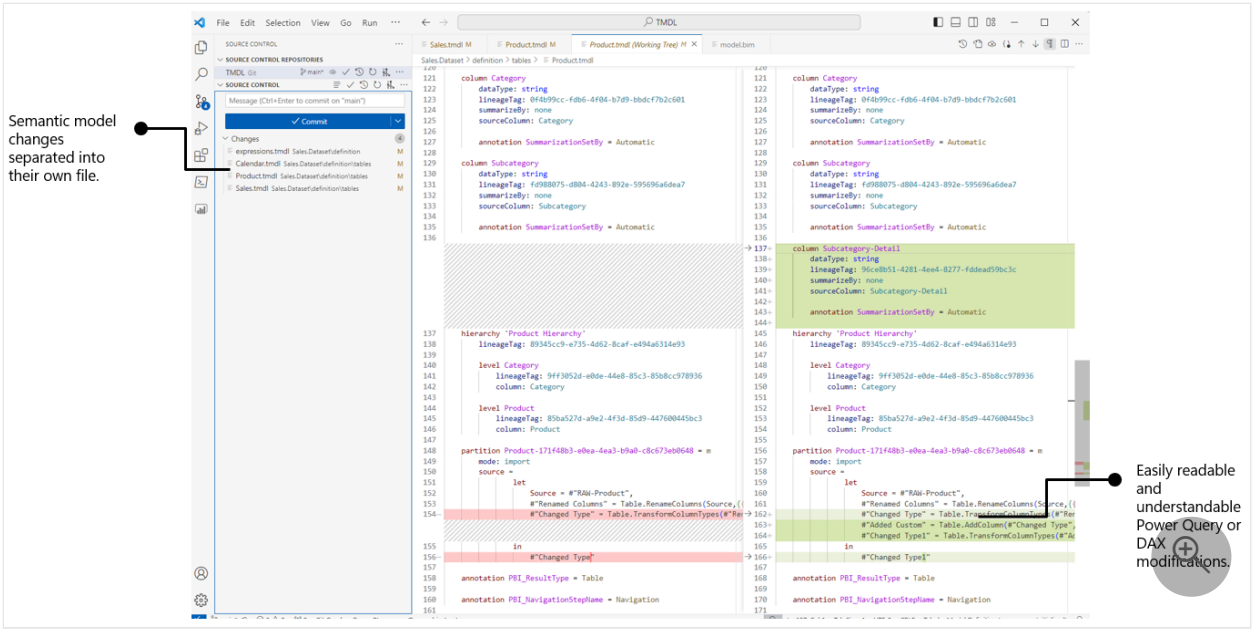
With the objective of providing a better source control and co-development experience, you can save your Power BI Project files (PBIP) using Tabular Model Definition Language (TMDL) as the semantic model file format.

Unlike Tabular Model Scripting Language (TMSL), TMDL has been designed from the ground up to be human-friendly, facilitating not only *readability* but also easy *editing* in any text editor. This represents a substantial enhancement for source control and *collaborative* development experiences.



Instead of a big JSON file like Tabular Model Scripting Language (TMSL), TMDL has a folder structure with separate files for each table, perspective, role, and culture. This makes it easier to work with others, and to understand the semantic model structure by

just looking at the folder and files. Ultimately, this leads to a great source control and co-development experience when dealing with git diff's and merge conflicts.



Learn more about TMDL here.

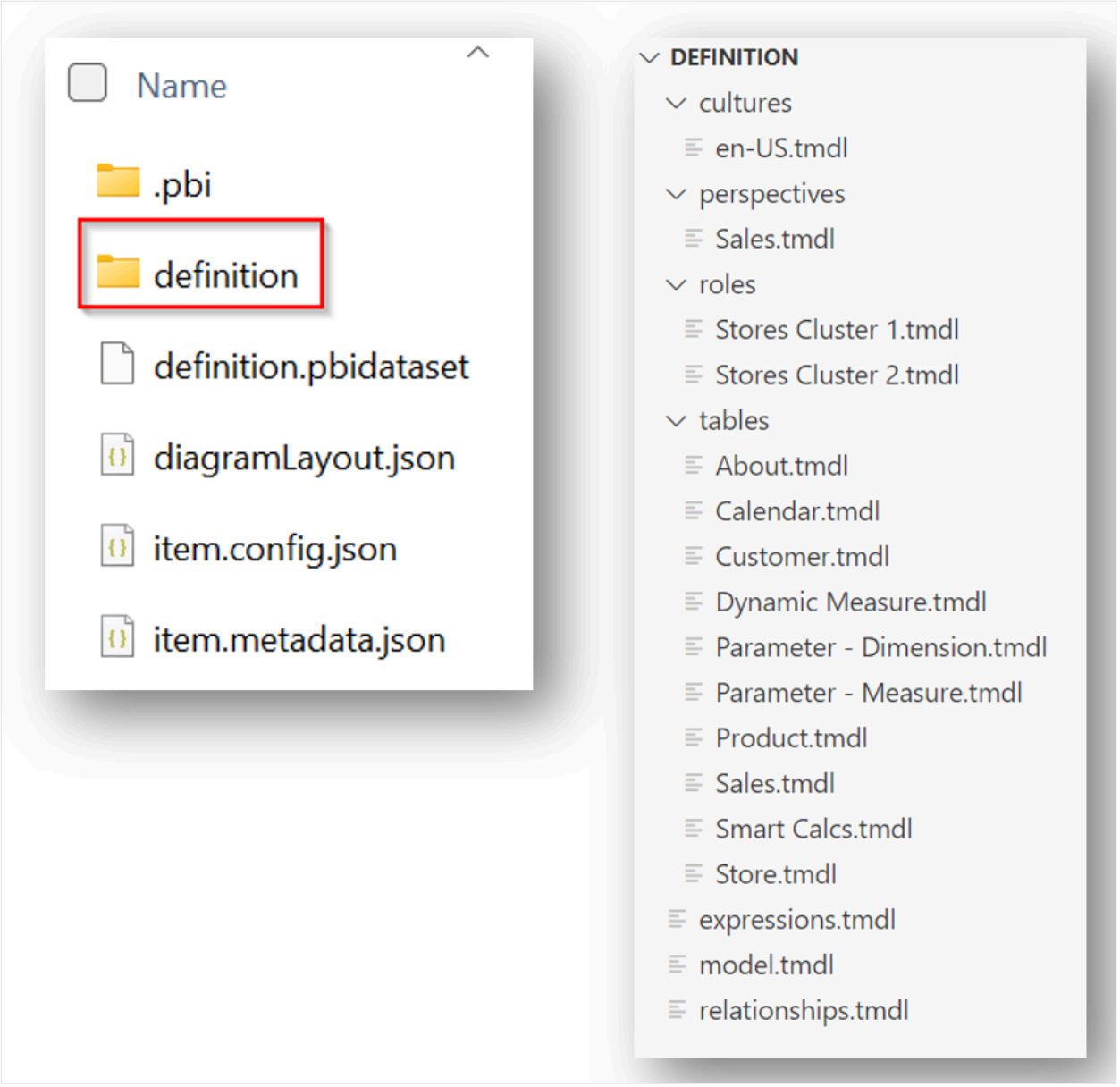
Enable TMDL format Preview feature

Saving as a Power BI Project using TMDL is currently in preview. Before using it, you must first enable it in Preview features:

Go to **File** > **Options and settings** > **Options** > **Preview features** and check the box next to **Store semantic model using TMDL format**.

Save as a project using TMDL

With the TMDL Preview feature enabled, when you save a project, your semantic model is saved as a TMDL folder named *\definition* inside of semantic model folder:



Learn more about the TMDL folder structure.

Convert existing PBIP to TMDL

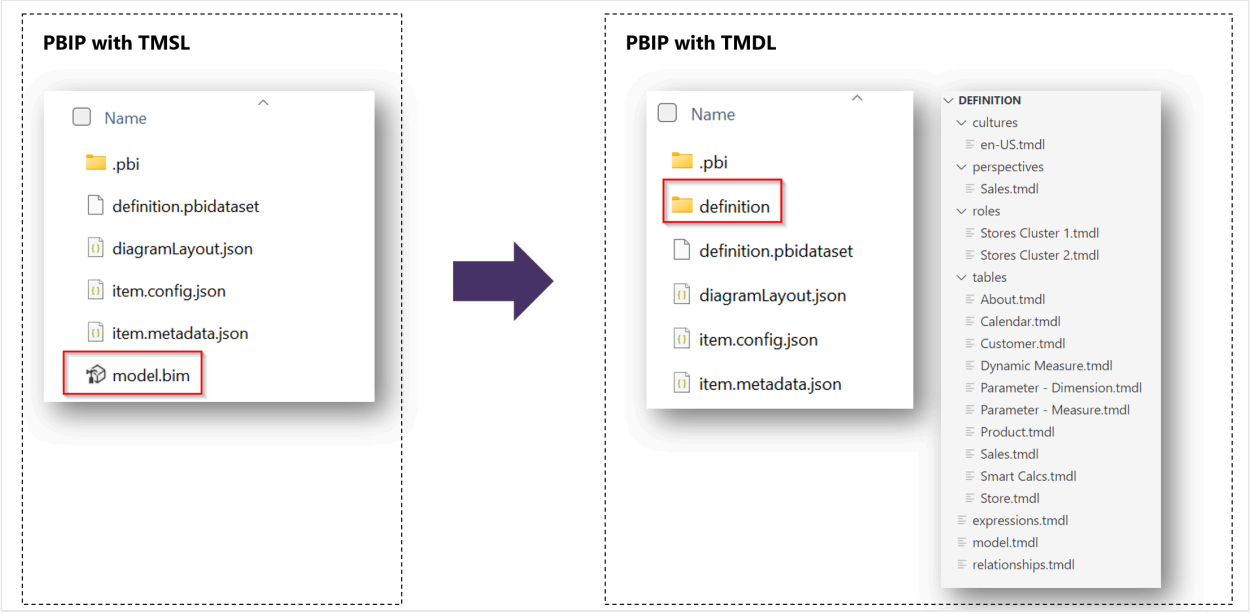
If you already have a PBIP using TMSL as semantic model format, you can convert it to TMDL as follows:

1. Open the PBIP in Power BI Desktop.
2. Ensure the Preview Feature is enabled.
3. **Save** the project. A prompt appears asking you to upgrade into TMDL.
4. Select **Upgrade**.

> **Important**

> Once you upgrade to TMDL, you can't revert back to TMSL. If you think you might want to revert back to TMSL, save a copy of your PBIP files first.

The existing Tabular Model Scripting Language (TMSL) file (*model.bim*) is replaced with a *\definition* folder containing the TMDL representation of the semantic model.

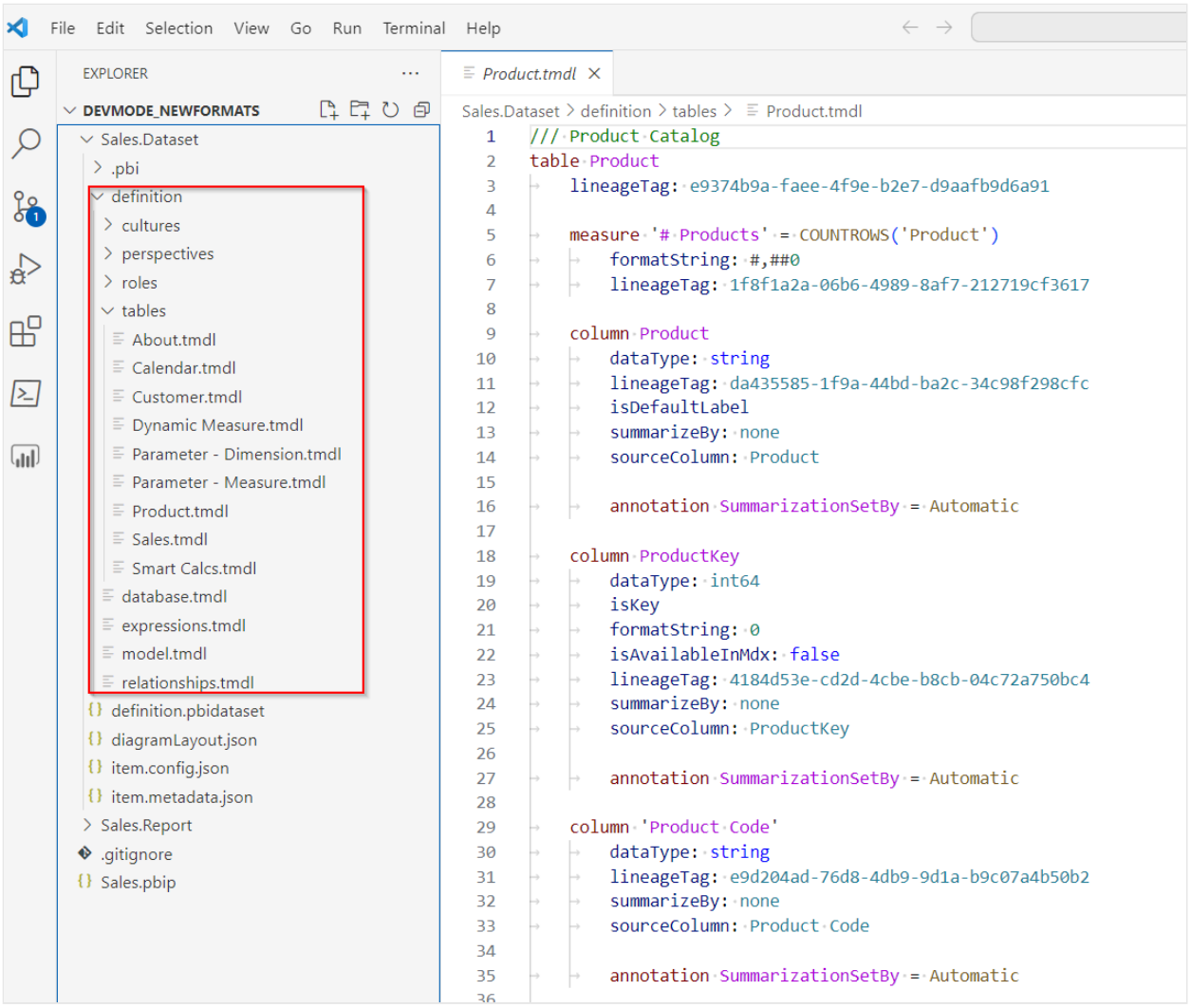


If you select to **Keep current** format, Desktop won't prompt again to upgrade.

Make external changes to TMDL files

For a better experience reading and editing your TMDL files, install the TMDL - Visual Studio Marketplace Microsoft VS Code extension.

Open the PBIP folder using VS Code and navigate to semantic model definition folder.

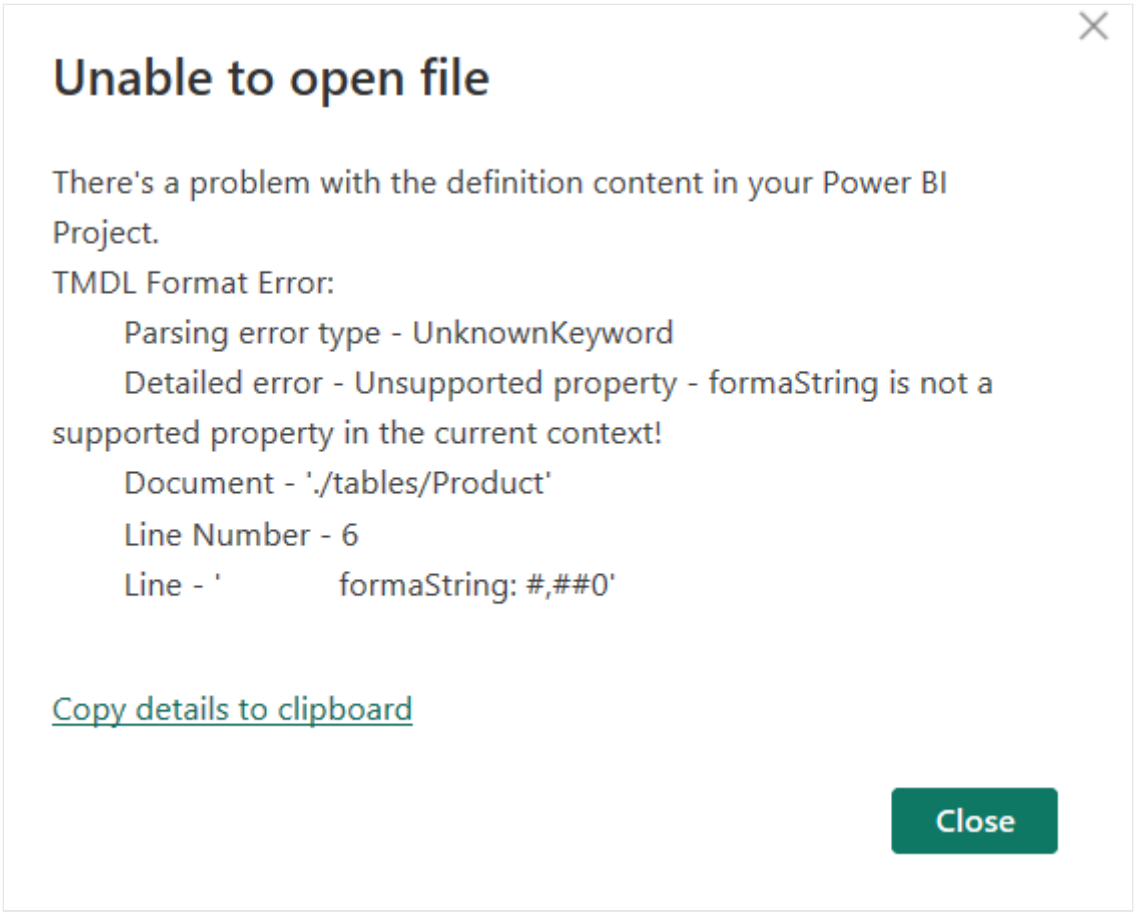


Power BI Desktop isn't aware of changes to project files made by other tools. Therefore, if you make any changes to open files outside of Power BI Desktop, you need to restart for those changes to be shown in Power BI Desktop.

Please refer here for supported write operations outside of Power BI Desktop.

TMDL Errors

If any invalid edits are made to the TMDL files, Power BI Desktop throws an error on open, with the location of the error:



TMDL considerations and limitations

During the Public Preview, Fabric Git Integration will still export the semantic model using TMSL by default. However, if the semantic model is imported into Fabric using Fabric Git Integration with TMDL format, then Fabric Git Integration will use TMDL format to export the semantic model definition to Git if there are any semantic model changes in the service.

> **ⓘ Important**
>
> If you import your semantic model using any other import method, such as **Power BI Desktop Publish**, Fabric Git Integration will switch back to the default TMSL format.

Related content

- Power BI Desktop project report folder
- Power BI Desktop projects
- Tabular Model Scripting Language (TMSL)

Feedback

Was this page helpful? Yes No

Provide product feedback | Ask the community

Power BI Desktop project report folder

Article • 08/13/2024

> **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\
- semanticModelDiagramLayout.json
- definition.pbir[1]
- mobileState.json
- report.json[2]
- definition\ folder[3]
- .platform

1 - This file is required.
2 - This file is required when saving to PBIR-Legacy format.
3 - This file is required when saving to PBIR format.

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, see 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 pbiviz 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 (pbiviz 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.

semanticModelDiagramLayout.json

Contains data model diagrams describing the structure of the semantic model 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 semantic model 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 semantic model 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 semantic model used in the report. The reference can be either:

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

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

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

Expand table

Property	Description
connectionString	The connection string referring to the remote semantic model.
pbiModelDatabaseName	The remote semantic model ID.
connectionType	Type of connection. For service remote semantic model, this value should be `pbiServiceXmlaStyleLive`.
pbiModelVirtualServerName	An internal property that should have the value, `sobe_wowvirtualserver`.

Example using `byConnection`:

JSON

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

When the semantic model and report share the same workspace, Fabric Git Integration always uses a `byPath` reference to the semantic model.

This file also specifies the supported report definition formats through the 'version' property.

Expand table

Version	Supported formats
1.0	Report definition must be stored as PBIR-Legacy in the report.json file.
4.0 or above	Report definition can be stored as PBIR-Legacy (report.json file) or PBIR (\definition folder).

For more information, see 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

This file contains the report definition in the Power BI Report Legacy format (PBIR-Legacy) and doesn't support external editing.

definition\ folder

This folder is only available if the Power BI project is saved using the Power BI enhanced report format (PBIR). It replaces the report.json file.

.platform

Fabric platform file that holds properties vital for establishing and maintaining the connection between Fabric items and Git.

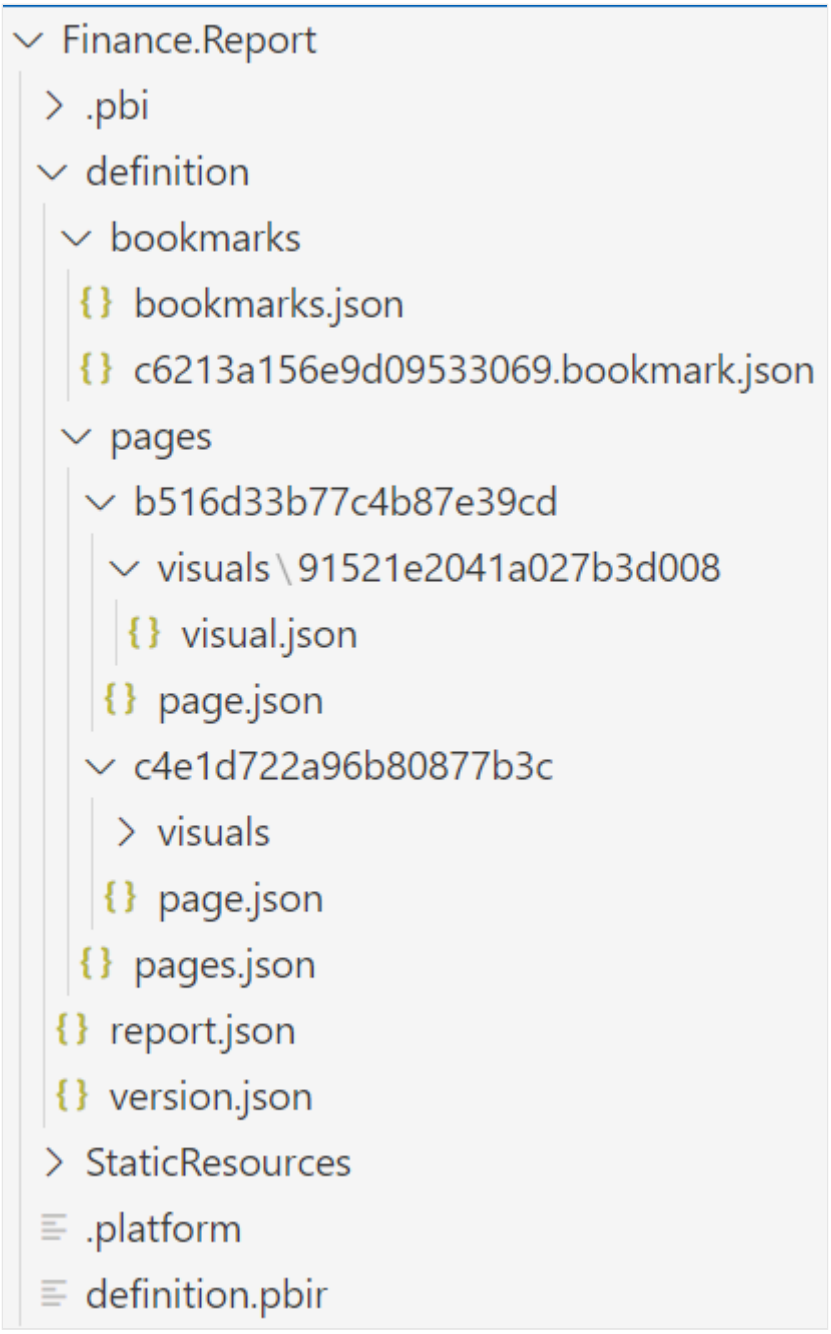
To learn more, see Git integration automatically generated system files.

PBIR format

> **ⓘ Important**
>
> Please consider all the PBIR **limitations** during the preview phase.

Saving your Power BI Project files (PBIP) using the Power BI Enhanced Report Format (PBIR) greatly improves change tracking and merge conflict resolution by using properly formatted JSON files.

Each page, visual, bookmark, etc., is organized into a separate, individual file within a folder structure. This format is ideal for co-development conflict resolution.



Unlike PBIR-Legacy (report.json), PBIR is a publicly documented format that supports modifications from non-Power BI applications. Each file has a public JSON schema, which not only documents the file but also lets code editors like Visual Studio Code perform syntax validation while editing.

Some of the possible scenarios now available with PBIR include:

- Copy pages/visuals/bookmarks between reports.
- Ensure consistency of a set of visuals across all pages, by copying & pasting the visual files.
- Easy find and replace across multiple reports files.
- Apply a batch edit across all visuals using a script (for example, hide visual level filters)

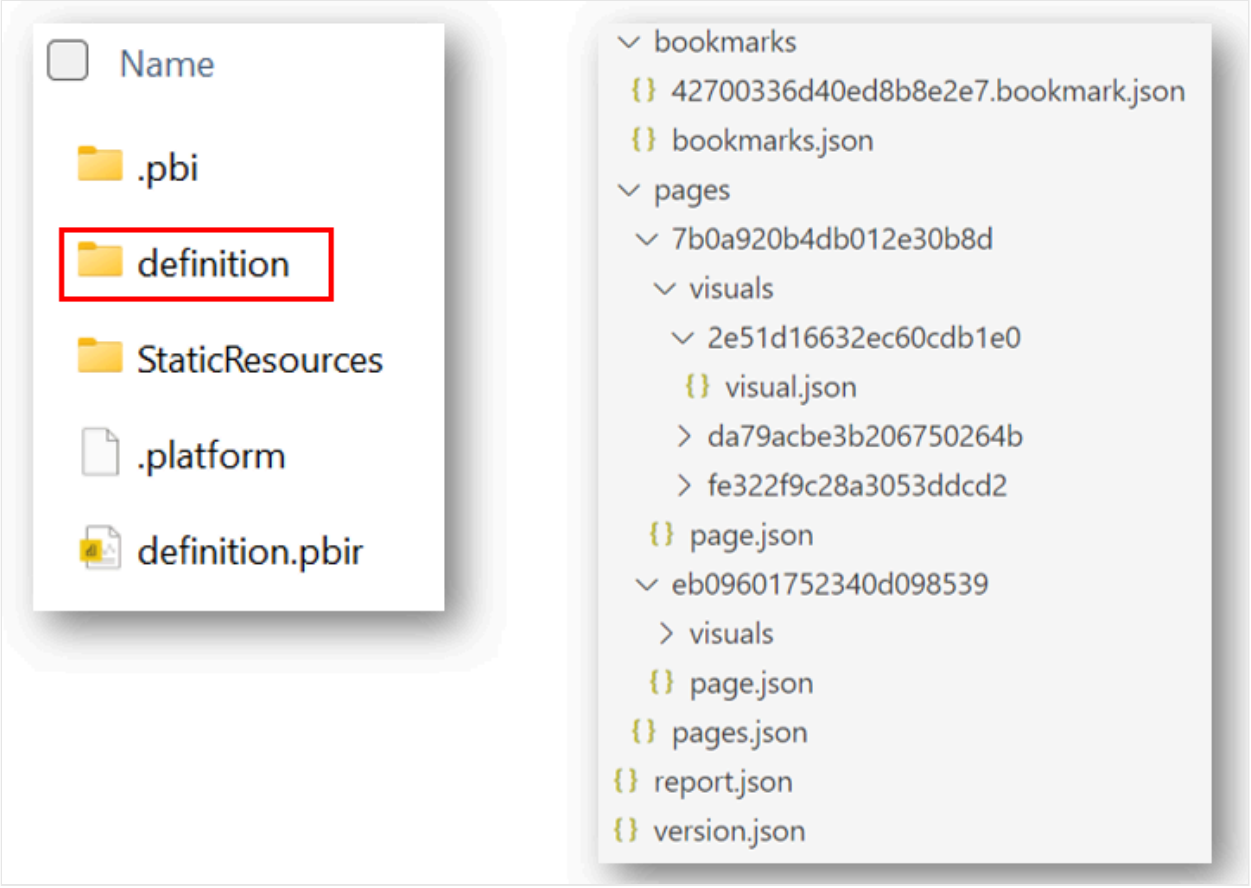
Enable PBIR format Preview feature

Saving as a Power BI Project using PBIR is currently in preview. Before using it, enable it in Power BI Desktop preview features:

Go to **File** > **Options and settings** > **Options** > **Preview features** and check the box next to **Store reports using enhanced metadata format (PBIR)**.

Save as a project using PBIR

With the PBIR Preview feature enabled, when you save a project, your report is saved within a folder named *definition* inside of report folder:



Learn more about the PBIR folder structure.

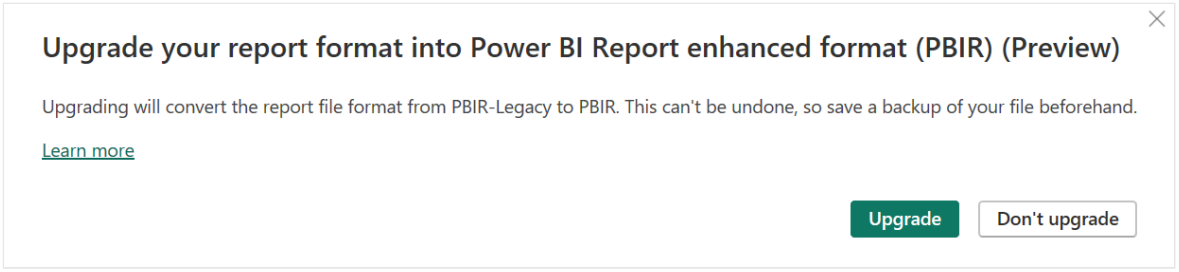
Convert existing PBIP to PBIR

If you already have a PBIP using PBIR-Legacy format, you can convert it to PBIR as follows:

1. Open the PBIP in Power BI Desktop.
2. Ensure the Preview Feature is enabled.

3. **Save** the project. A prompt appears asking you to upgrade into PBIR.

4. Select **Upgrade**.



> **Important**
>
> Once you upgrade to PBIR, you can't revert back to PBIR-Legacy. If you think you might want to revert back to PBIR-Legacy, save a copy of your PBIP files first.

The existing PBIR-Legacy file (*report.json*) is replaced with a *\definition* folder containing the PBIR representation of the report.

If you select to **Keep current** format, Desktop don't prompt again to upgrade.

Publish a PBIR report to service

While in the **preview** phase, the only way to publish a report with the PBIR format is through Fabric Git Integration. This involves connecting the workspace to a Git repository and pushing the PBIR report to it, which can then be synchronized with the service workspace at a later stage.

If you wish to convert an existing report to PBIR in the service, follow these steps:

1. Connect your workspace to Git.
2. Clone the Git repository to your local file system.
3. Open the report in Power BI Desktop by opening the `definition.pbir` file.
4. Save the report and choose to upgrade to PBIR.
5. Commit and sync the changes to Git.
6. Update the workspace with the latest changes from Git.

PBIR folder and files

The report definition is stored inside the `definition\` folder with the following structure:

```
Markdown
```

```
├── bookmarks\
│   ├── [bookmarkName].bookmark.json
│   └── bookmarks.json
├── pages\
│   ├── [pageName]\
│   │   ├── \visuals
│   │   │   ├── [visualName]\
│   │   │   │   ├── mobile.json
│   │   │   └   └── visual.json
│   │   └── page.json
│   └── pages.json
├── version.json
├── reportExtensions.json
└── report.json
```

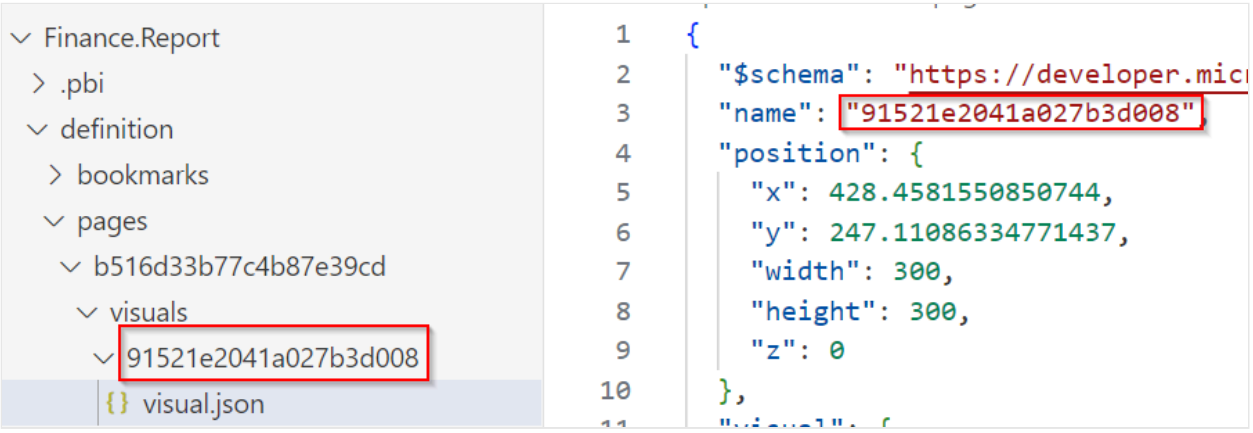
Expand table

File/Folder	Required	Description
bookmarks\	No	Folder holding all bookmark files of the report.
── [bookmarkName].bookmark.json	No	Bookmark metadata, such as target visuals and filters. More information at schema.
── bookmarks.json	No	Bookmarks metadata, such as bookmark order and groups. More information at schema.
pages\	Yes	Folder holding all pages of the report.
── [pageName]\	Yes	One folder per page.
──── visuals\	No	Folder holding all visuals of the page.
────── [visualName]\	No	One folder per visual.
──────── mobile.json	No	Visual mobile layout metadata, such as mobile position and formatting. More information at schema.
──────── visual.json	Yes	Visual metadata, such as position and formatting, query. More information at schema.
──── page.json	Yes	Page metadata, such as page level filters and formatting. More information at schema.
── pages.json	No	Pages metadata, such as page order and active page.

File/Folder	Required	Description
		More information at schema.
version.json	Yes	PBIR file version, among other factors, determines the required files to be loaded. More information at schema
reportExtensions.json	No	Report extensions, such as report level measures. More information at schema
report.json	Yes	Report metadata, such as report level filters and formatting. More information at schema

PBIR naming convention

All names inside the square brackets ([]) in the preceding table follow a default naming convention but can be renamed to more user-friendly names. By default, pages, visuals, and bookmarks use their report object name as their file or folder name. These object names are initially a 20-character unique identifier, such as '90c2e07d8e84e7d5c026'.



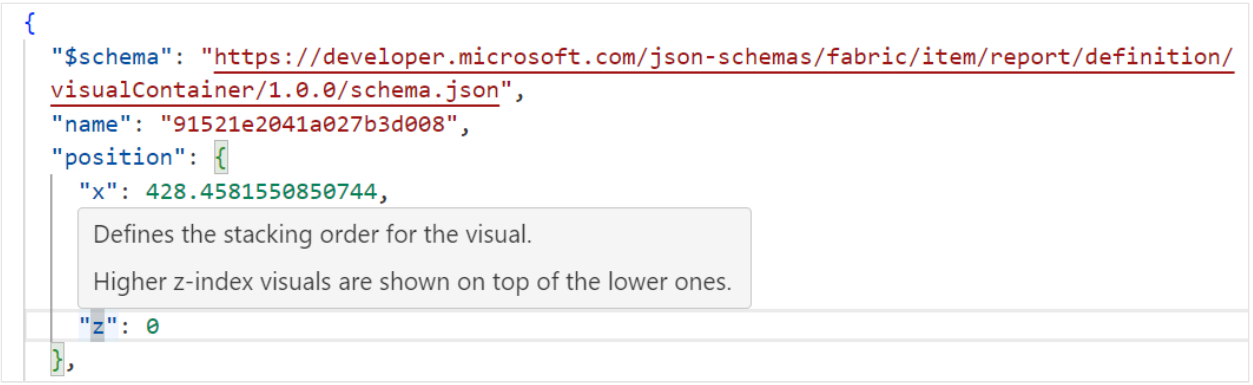
Renaming the 'name' property within each JSON file is supported but might break external references both inside and outside the report. The object name and/or file/folder name must consist of one or more word characters (letters, digits, underscores) or hyphens.

After renaming any PBIR files or folders, you must restart Power BI Desktop. Upon restart, Power BI Desktop will preserve the original file or folder names when saving.

PBIR Json Schemas

Each PBIR JSON file includes a JSON schema declaration at the top of the document. This schema URL is publicly accessible and can be used to learn more about the

available properties and objects for each file. Additionally, it provides built-in IntelliSense and validation when editing with code editors like Visual Studio Code.



The schema URL also defines the version of the document, which is expected to change as the report definition evolves.

All the JSON schemas are published here.

PBIR annotations

You can include annotations as name-value pairs within the report definition for each `visual`, `page` and `report`. While Power BI Desktop will ignore these annotations, they can be valuable for external applications like scripts.

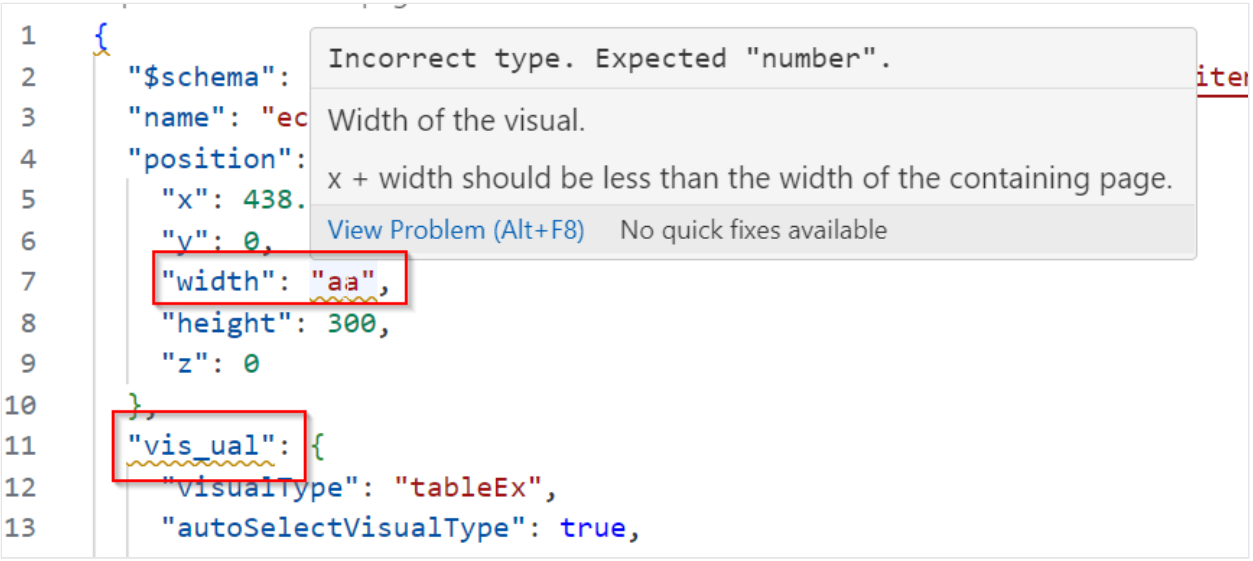
For instance, you could specify the defaultPage for the report at the `report.json` file, which can then be utilized by a deployment script.

JSON

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/fabric/item/report/definition/report/1.0.0/schema.json",
  "themeCollection": {
    "baseTheme": {
      "name": "CY24SU06",
      "reportVersionAtImport": "5.55",
      "type": "SharedResources"
    }
  },
  ...
  "annotations": [
    {
      "name": "defaultPage",
      "value": "c2d9b4b1487b2eb30e98"
    }
  ]
}
```

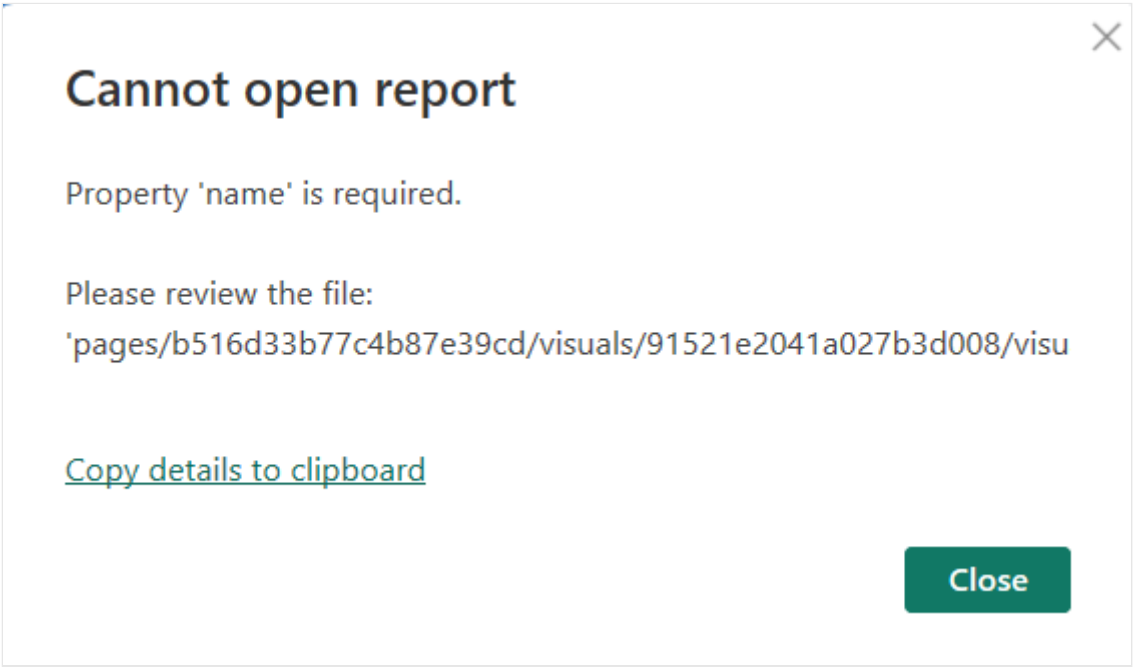
External changes to PBIR files

You can edit the PBIR JSON files using a code editor like Visual Studio Code or an external tool, as long as the file obey the JSON schema. Using a wrong property name or type can be easily detected directly in Visual Studio Code:



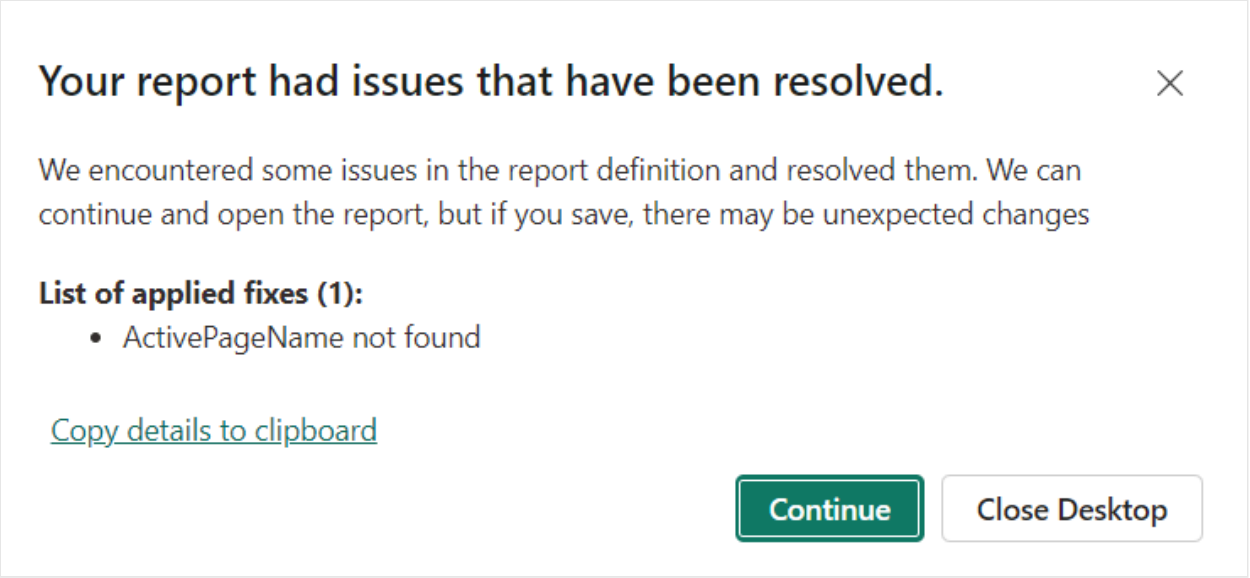
External changes to PBIR content might result in errors when reopening the files in Power BI Desktop. These errors can be of two types:

Blocking errors prevent Power BI Desktop from opening the report. These errors help identify the issue and the offending file that must be fixed before reopening:



Errors such as an invalid schema or missing required properties are considered blocking errors. Those errors can be easily identified by opening the file in Visual Studio Code and inspecting the schema errors.

Non-blocking errors don't prevent Power BI Desktop from opening the report and are automatically resolved.



Errors such as an invalid *activePageName* configuration are examples of nonblocking errors that are automatically fixed. The warning is necessary to give you the chance to avoid saving the report with the autofix, by that preventing any potential loss of work.

Common PBIR errors

Scenario: *After rename visual or page folder names, my visual or page no longer appears when opening the report.*

Solution: Verify whether the name complies with the naming convention. If it doesn't, Power BI Desktop ignores the file, or folder and treats it as private user files.

Scenario: *New report objects are named differently from others. For example, most page folders are named 'ReportSection0e71dafbc949c0853608', while a few are named '1b3c2ab12b603618070b'.*

Solution: PBIR adopted a new naming convention for every object, but it only applies to new objects. When you save an existing report as PBIP, the current names must be preserved to prevent breaking references. If you want consistency, a script a batch rename is allowed.

Scenario: *I copied a bookmark file, and upon saving, most of the bookmark configuration was deleted.*

Solution: This behavior is intentional, report bookmarks capture the state of a report page along with all its visuals. Since the captured state originates from another report page with different visuals, any invalid visuals are removed from the bookmark

configuration. If you also copy the dependent visuals and page, the bookmark maintain its configuration.

Scenario: *I copied a page folder from another report and encountered an error stating, "Values for the 'pageBinding.name' property must be unique."*

Solution: The pageBinding object is necessary to support drillthrough and page tooltips. Since they might be referenced by other pages, the name must be unique within the report. On the newly copied page, assign a unique value to resolve the error. After June 2024, this situation is no longer an issue because the pageBinding name is a GUID by default.

PBIR considerations and limitations

PBIR is currently in **preview**. Keep the following in mind:

- Service limitations
  - Can't be included in Power BI Apps.
  - Can't be downloaded as PBIX.
  - Can't be deployed with deployment pipelines.
  - Can't be saved as a copy.
  - Can't be published from Power BI Desktop.
  - Can't be uploaded to workspace as PBIX.
- Large reports with more than 500 files experience authoring performance issues (report viewing isn't affected), including:
  - Saving in Power BI Desktop
  - Synchronization in Fabric Git Integration
- Once a report is converted from PBIR-Legacy to PBIR, it isn't possible to roll it back.
- Converting a PBIP file to a PBIX file using the "Save As" feature embeds the PBIR report within the PBIX file, carrying over all PBIR limitations to the PBIX.

PBIR size limitations enforced by the service:

- 1,000 max pages per report.
- 300 max visuals per page.
- 5 mb max for each bookmark file.
- 1 mb max for each file.
- 1,000 max resource package files per report.
- 300 mb max size for all resource package files.
- 20 mb max size of all report files.

During the Public Preview, Fabric Git Integration and Fabric REST APIs continue to use PBIR-Legacy (report.json) when exporting the report definitions. However, if the report is

imported into Fabric using PBIR format, then both features start exporting the report definition using PBIR format.

Related content

- Power BI Desktop project semantic model folder
- Power BI Desktop projects

Feedback

Power BI Desktop projects Git integration

Article • 08/13/2024

> **ⓘ 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:

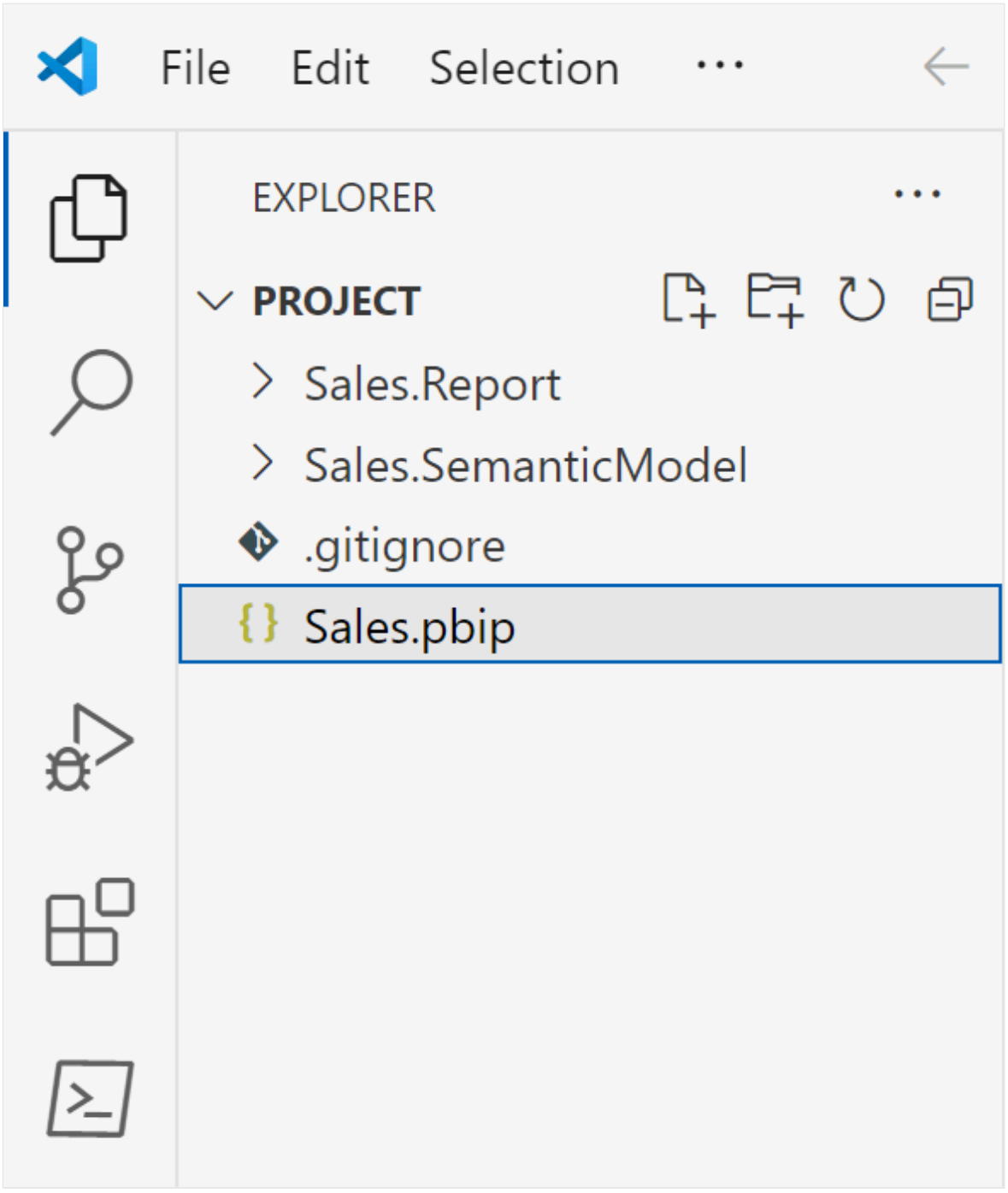
- Backup and version your work.
- Revert to previous states.
- Collaborate with others or work alone using Git branches.
- Use the capabilities of familiar source control tools, like Azure DevOps.

Prerequisites

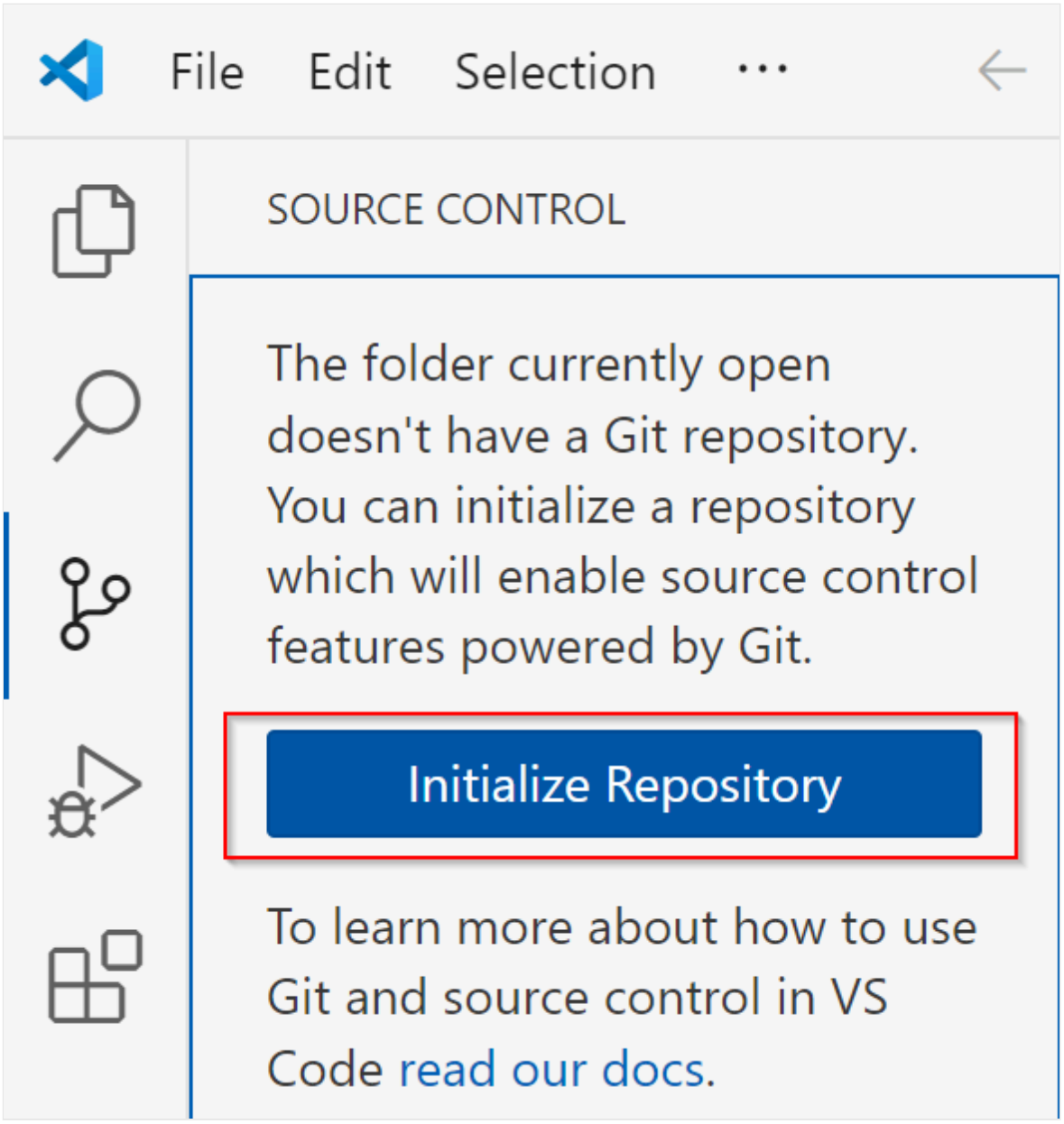
- Familiarity with Git. See Git and GitHub learning resources.
- Download and install Git.
- Download and install VS Code development environment. It has native integration with Git. To learn more, see Using Git source control in VS Code.

Create a local Git repo using VS Code

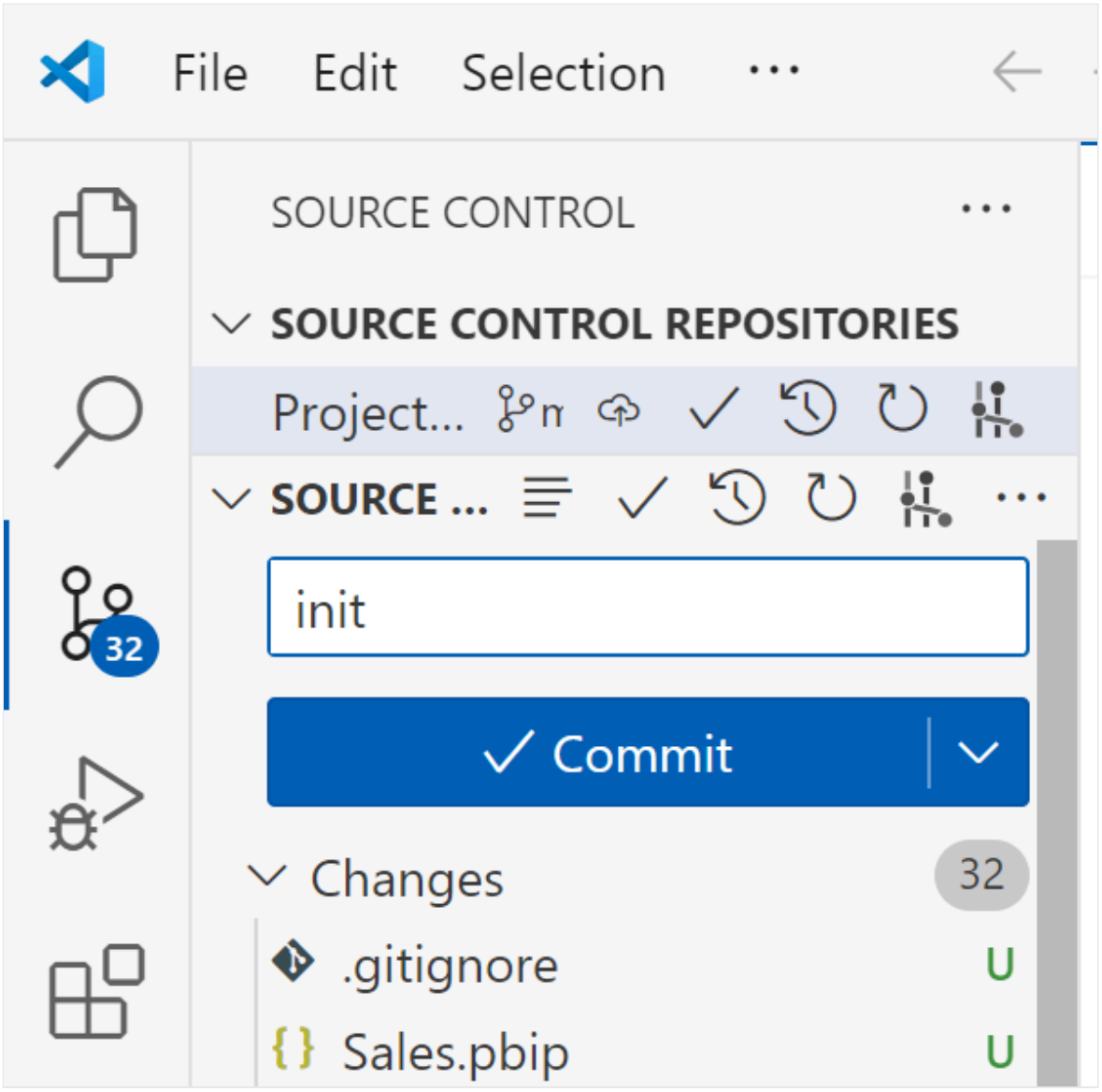
1. In VS Code, open a Power BI Desktop project folder:



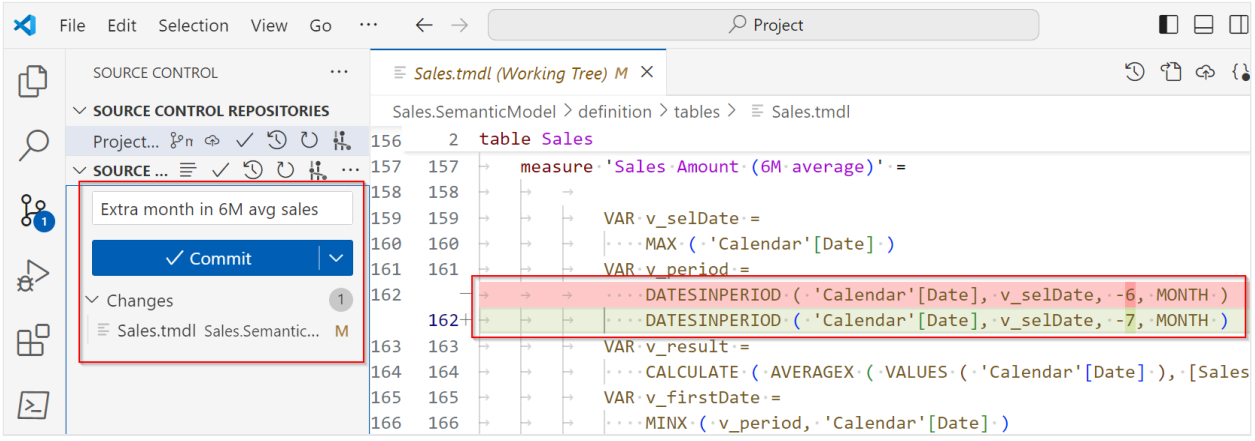
2. Initialize a Git repository by selecting **Source Control** > **Initialize Repository**:



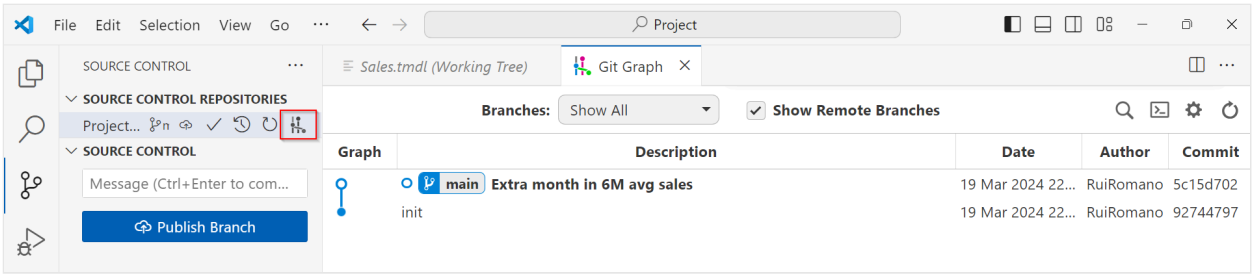
3. Do an initial Commit and enter a message:



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.



Related content

- Power BI Desktop projects Azure DevOps integration
- Power BI Desktop project semantic model folder
- Power BI Desktop project report folder

Feedback

Was this page helpful? Yes No

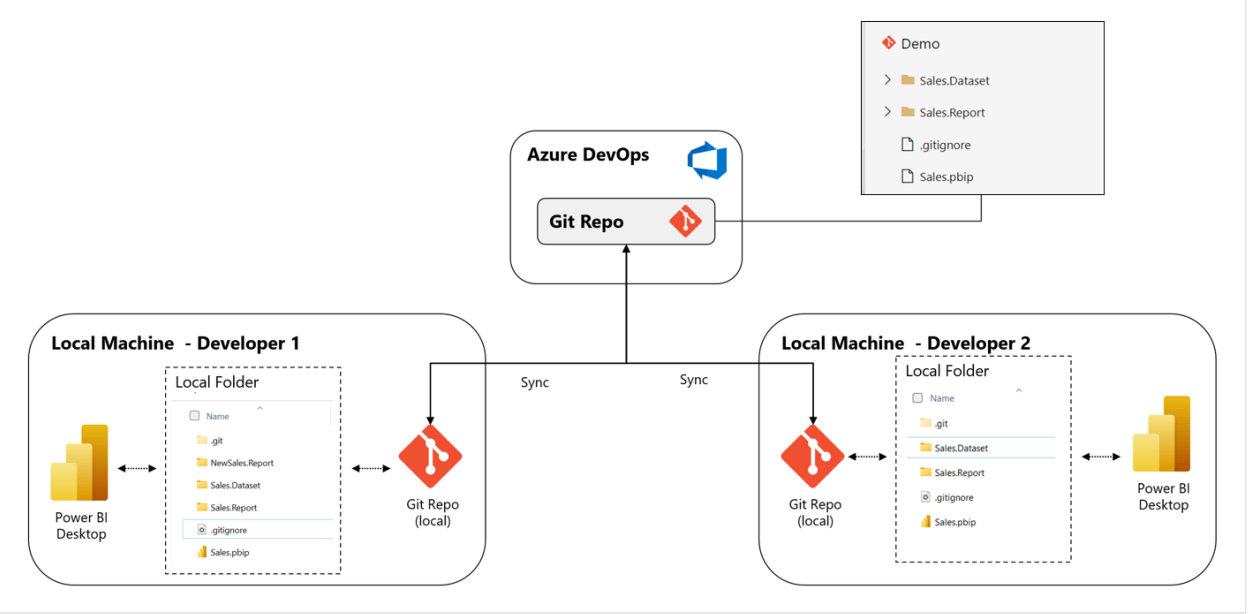
Provide product feedback | Ask the community

Power BI Desktop projects Azure DevOps integration

Article • 08/19/2024

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

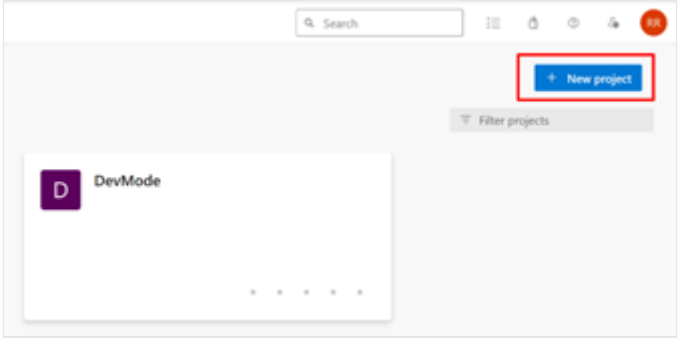
Microsoft Power BI Desktop projects enable developer collaboration by connecting your local Git repo to a remote Git host, like Azure DevOps.



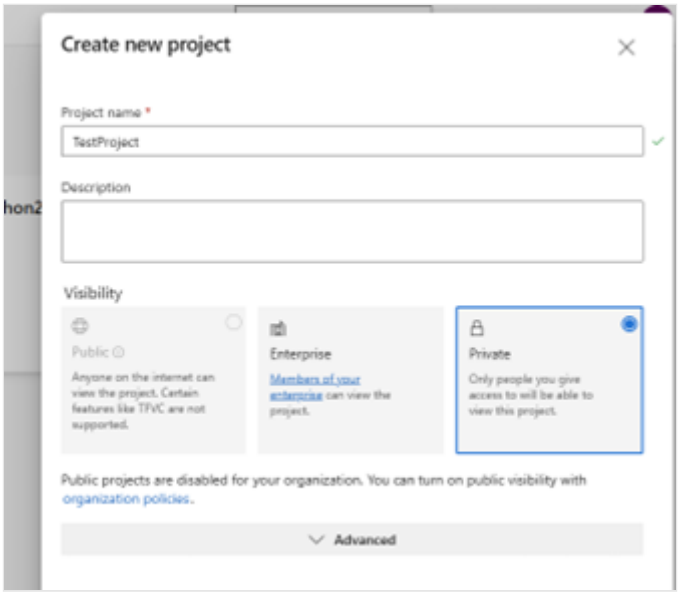
Create a Git repo in Azure DevOps

1. In Azure DevOps, select an existing organization, or create a new one.

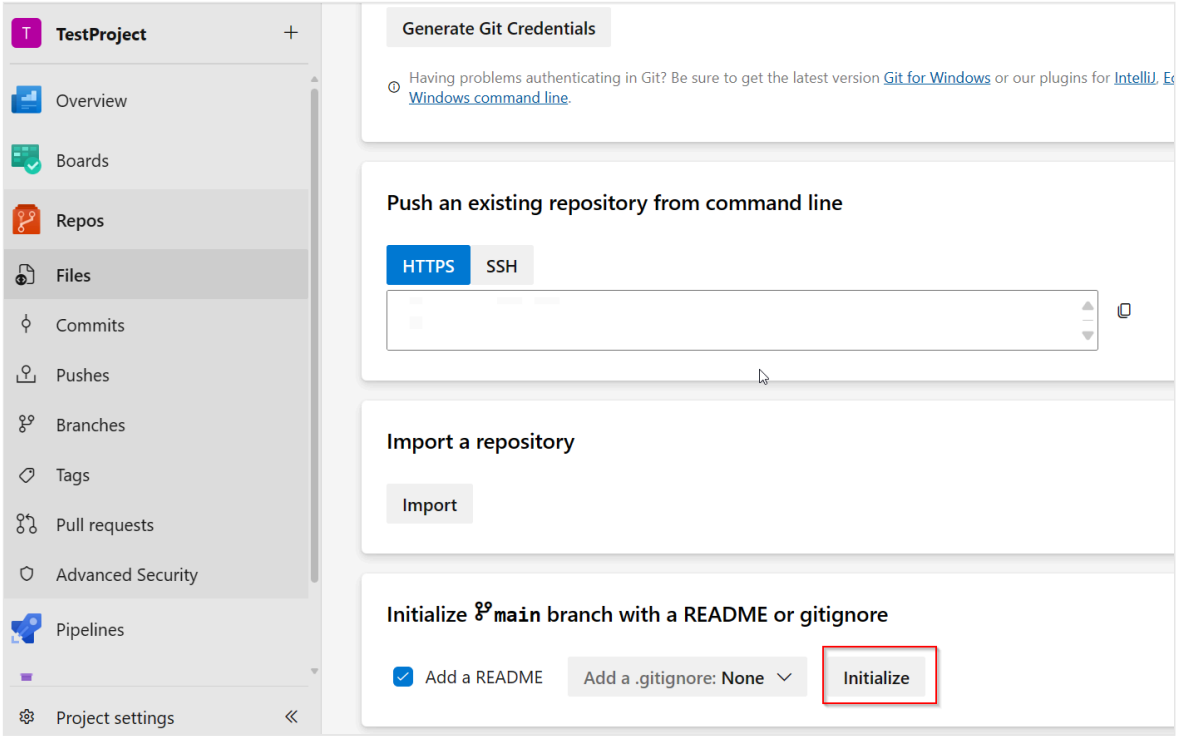
2. Create a new Project within the organization:



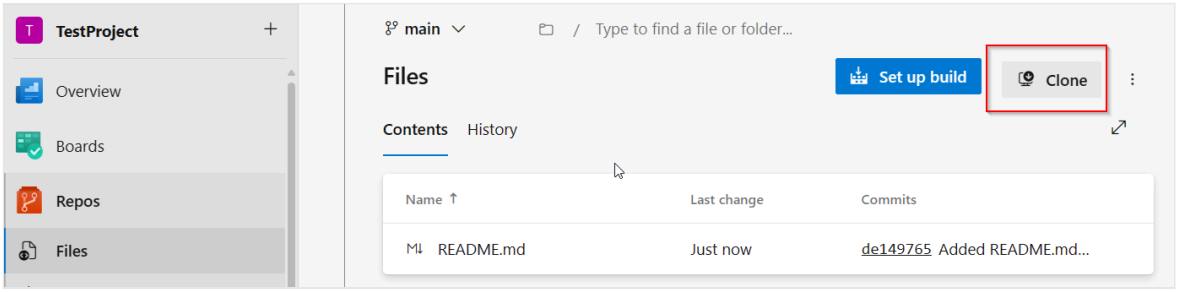
3. Enter your project details.



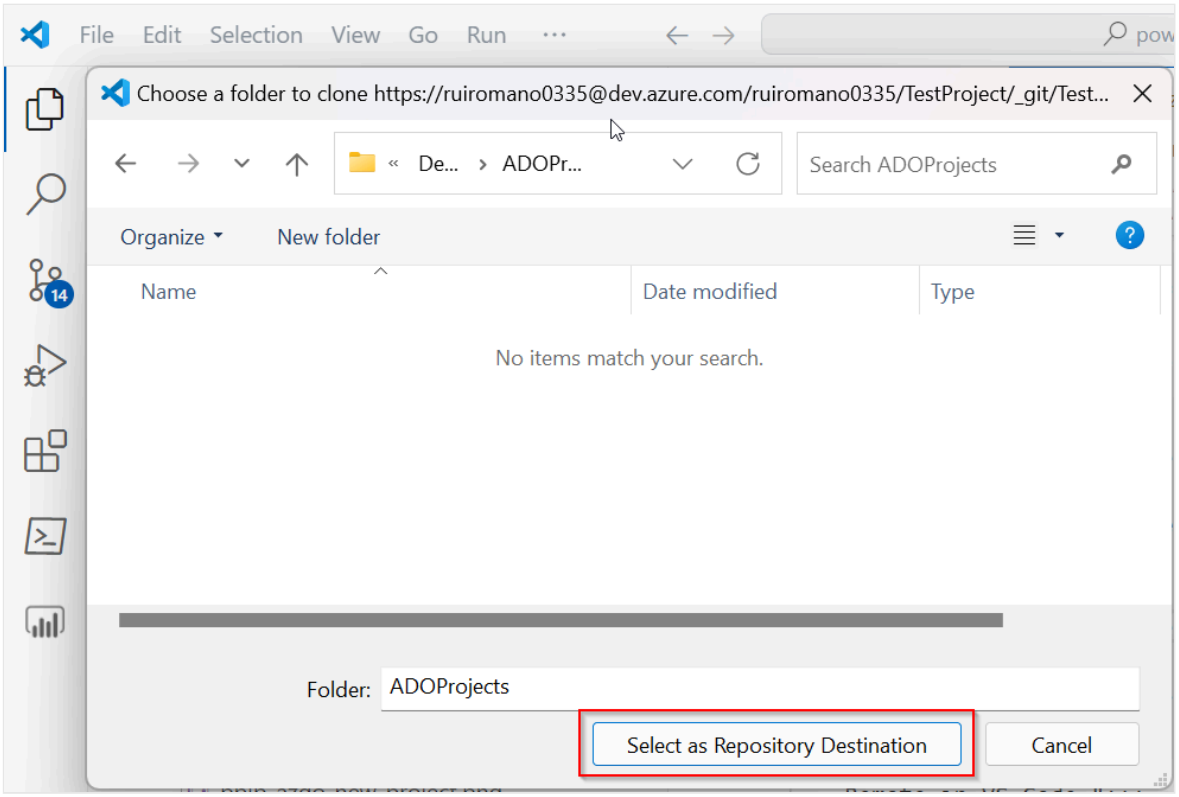
4. Select **Repos** > **Files**, and then select 'Initialize' to create an empty branch:

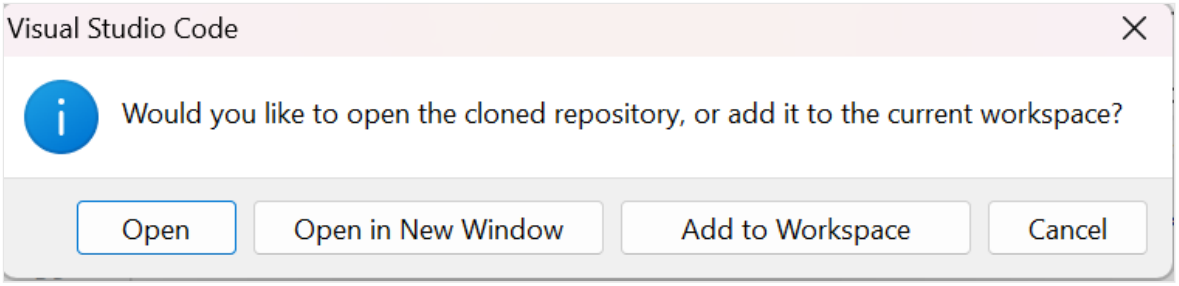


5. In **Repos** > **Files**, select **Clone** > **Clone in VS Code**

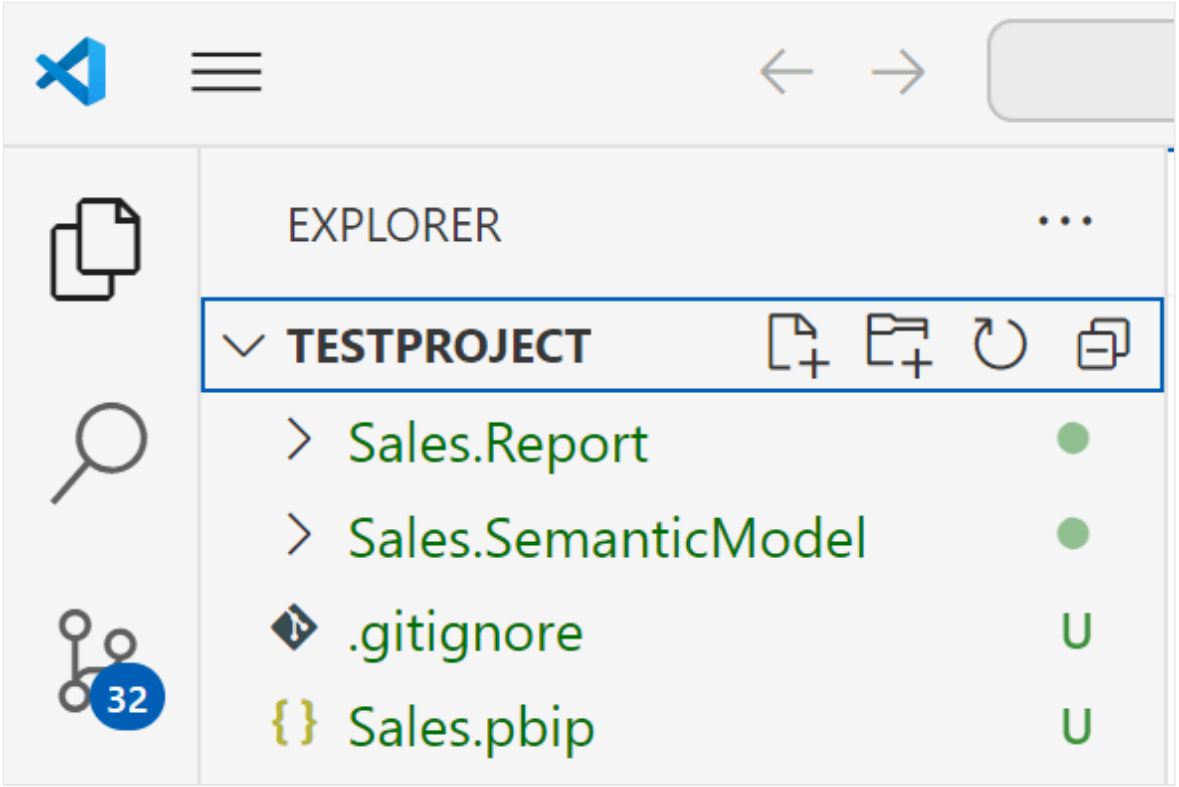


6. In Visual Studio Code (VS Code) > pick a folder and open

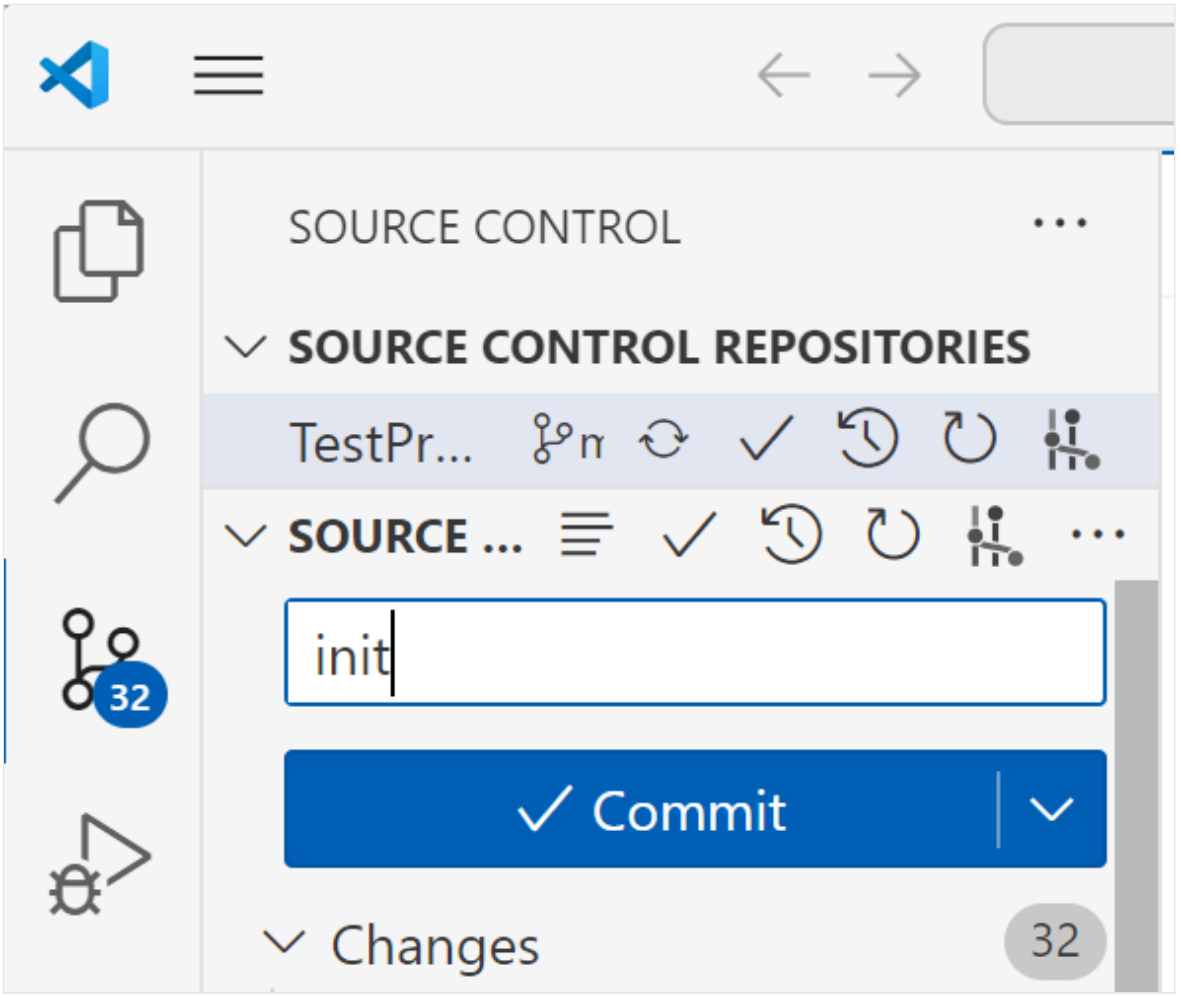


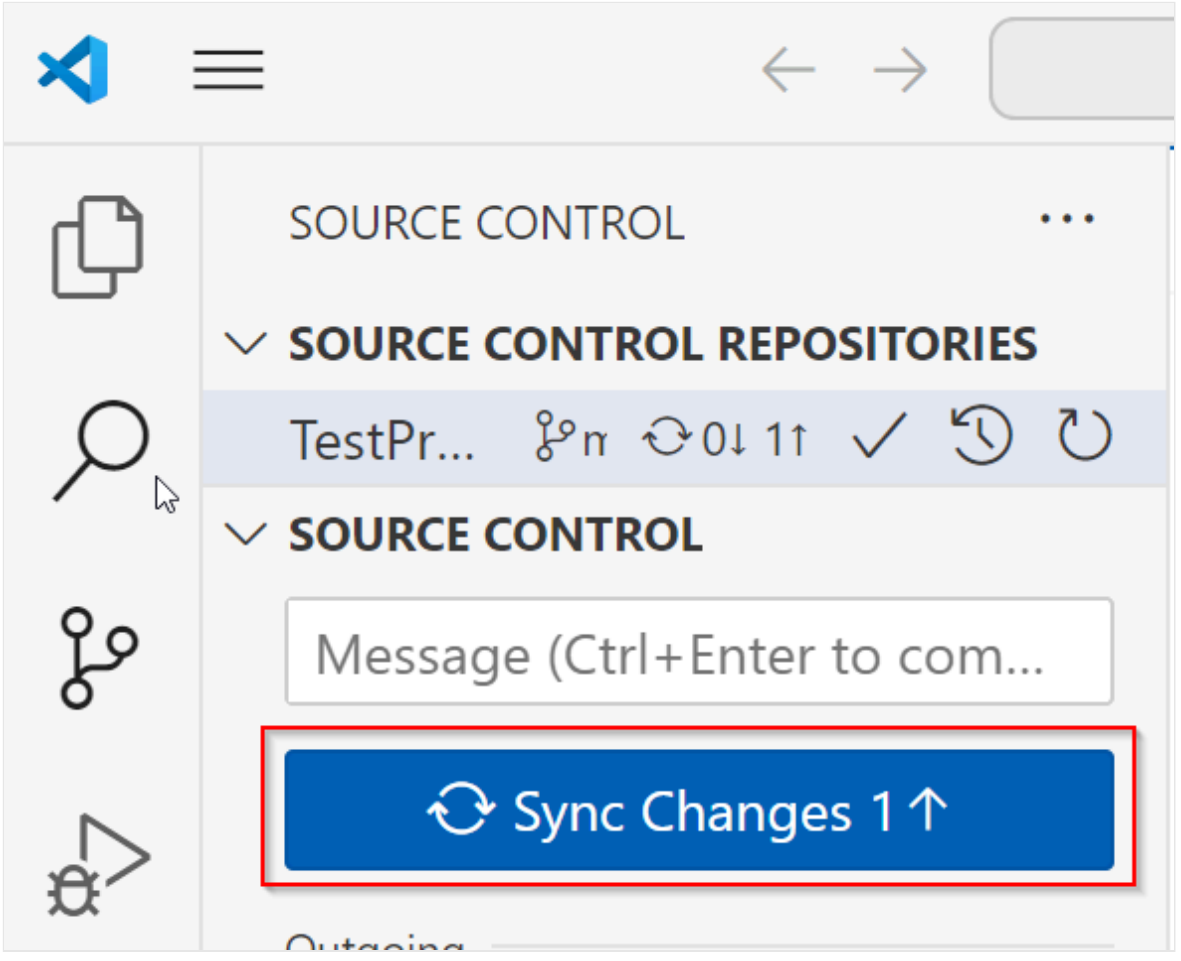


7. Save your Power BI Project files to the selected folder

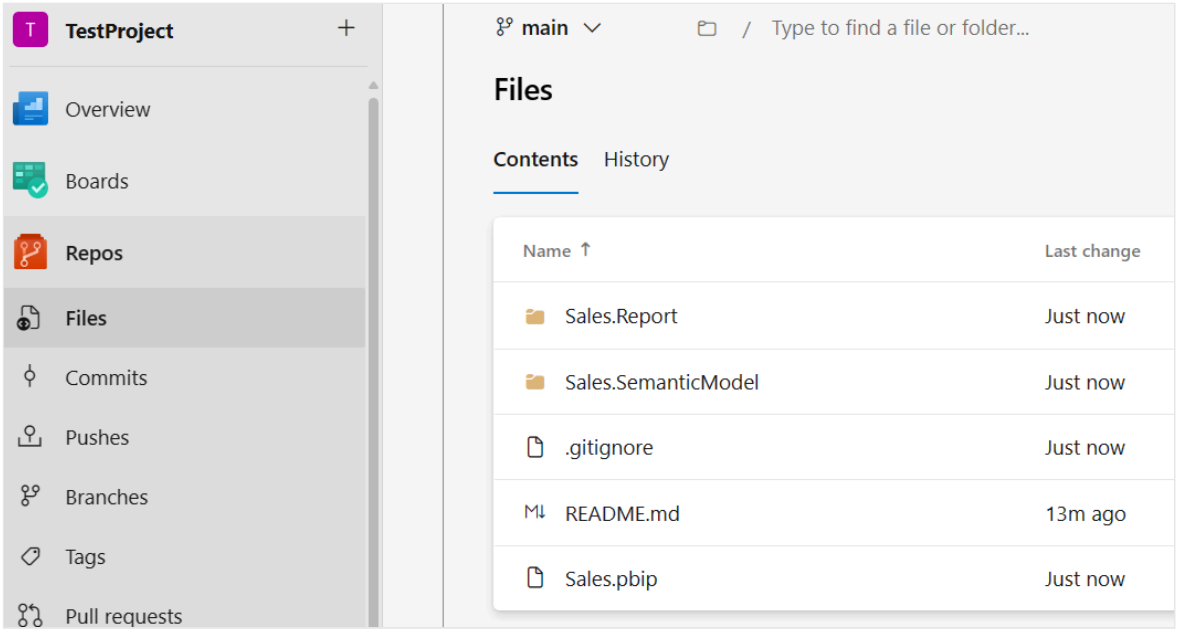


8. Commit and sync changes





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



As you can see, with Azure DevOps integration, you can have multiple developers working on the same Power BI project as long as they're 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.

Related content

- Power BI Desktop projects Git integration
- Power BI Desktop projects

Feedback

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

Article • 12/09/2024

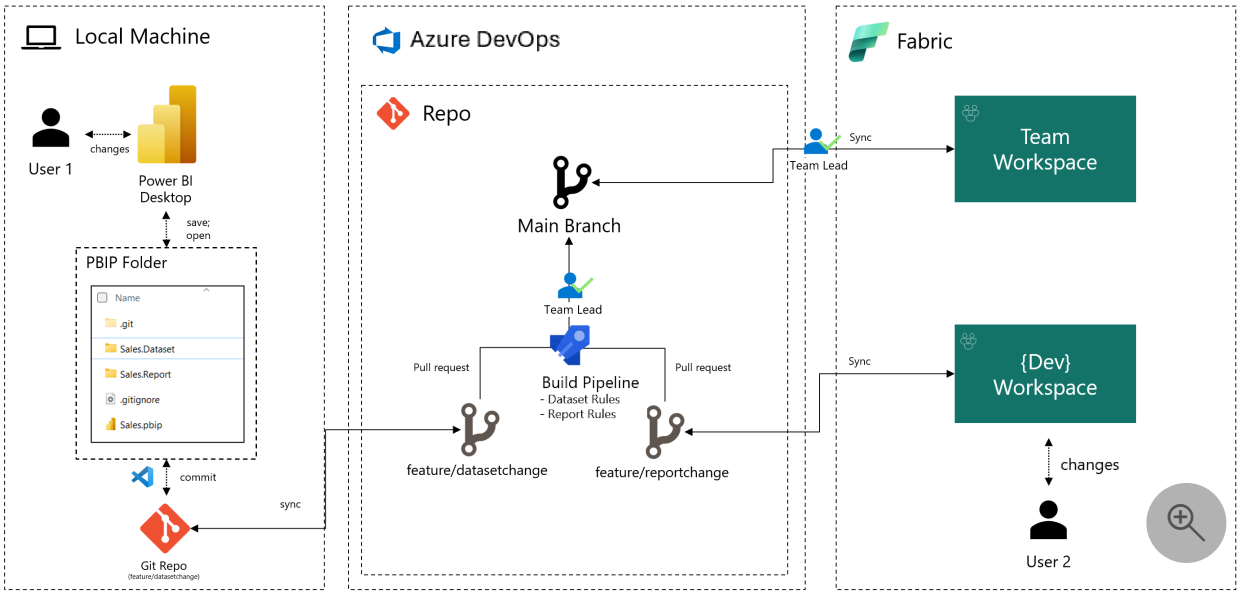
Combining Fabric Git Integration with Azure DevOps, enables you to connect a workspace to a branch in an Azure DevOps repository and automatically synchronizes between them.

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 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 pipeline to validate development quality. The pipeline 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

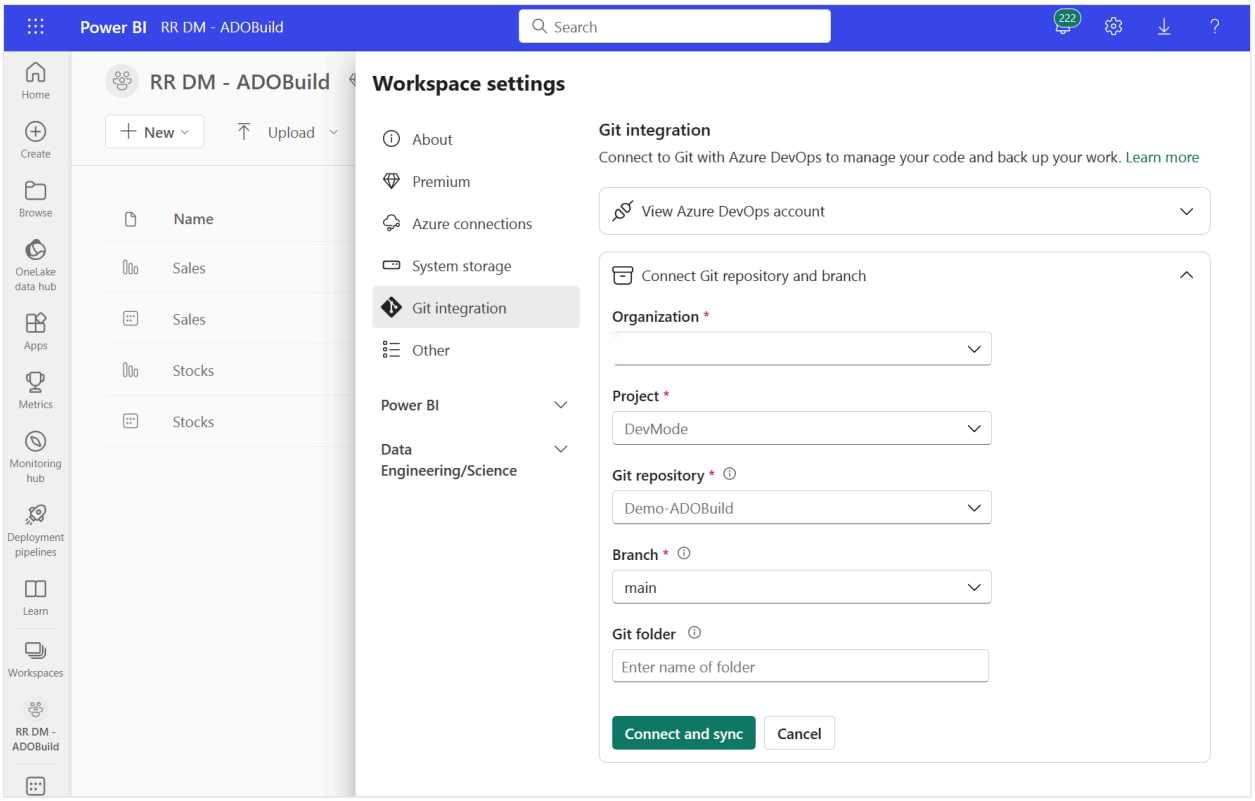
3. The team lead reviews the Pull Requests and synchronizes the changes to the team workspace using Fabric Git.

4. The Pull Request triggers the Azure DevOps pipeline to inspect the semantic model and report development quality.

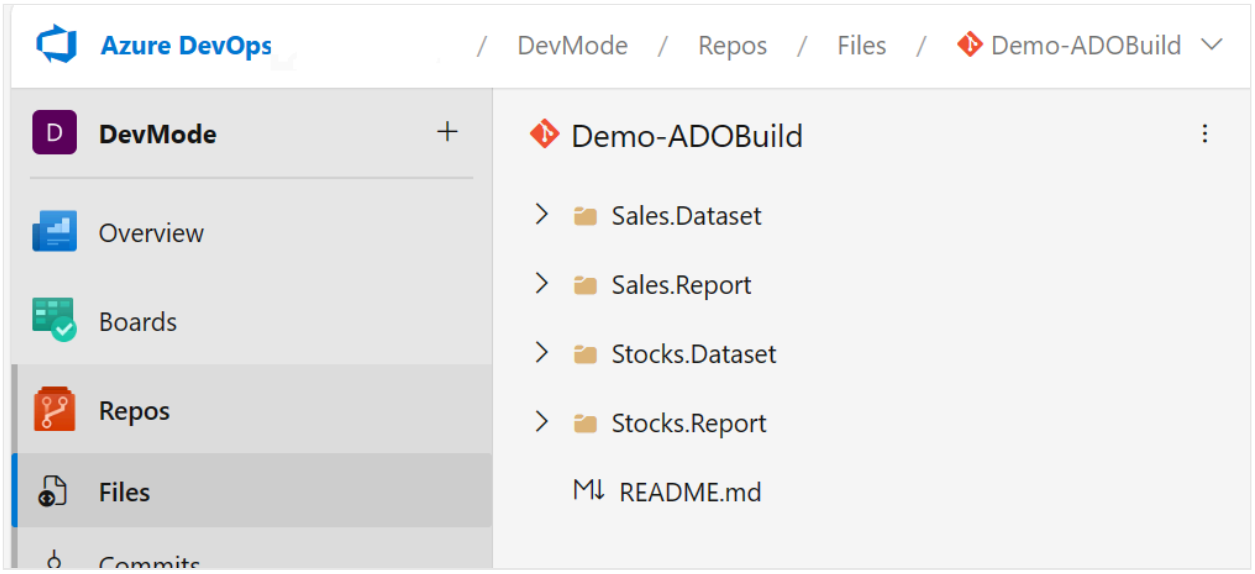
> **Note**
>
> In this example, the pipeline uses two open-source community tools that enable a developer to apply (customizable) best practice rules to the metadata of semantic models and reports within a Power BI Project folder:
>
> - Tabular Editor and Best Practice Rules
> - PBI Inspector
>
> An 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, 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 a Fabric Workspace. It's important to note that the referred community tools are developed by third-party contributors, and Microsoft doesn't offer support or documentation for them.

Step 1 - Connect Fabric workspace to Azure DevOps

Connect your Fabric workspace to Azure DevOps:



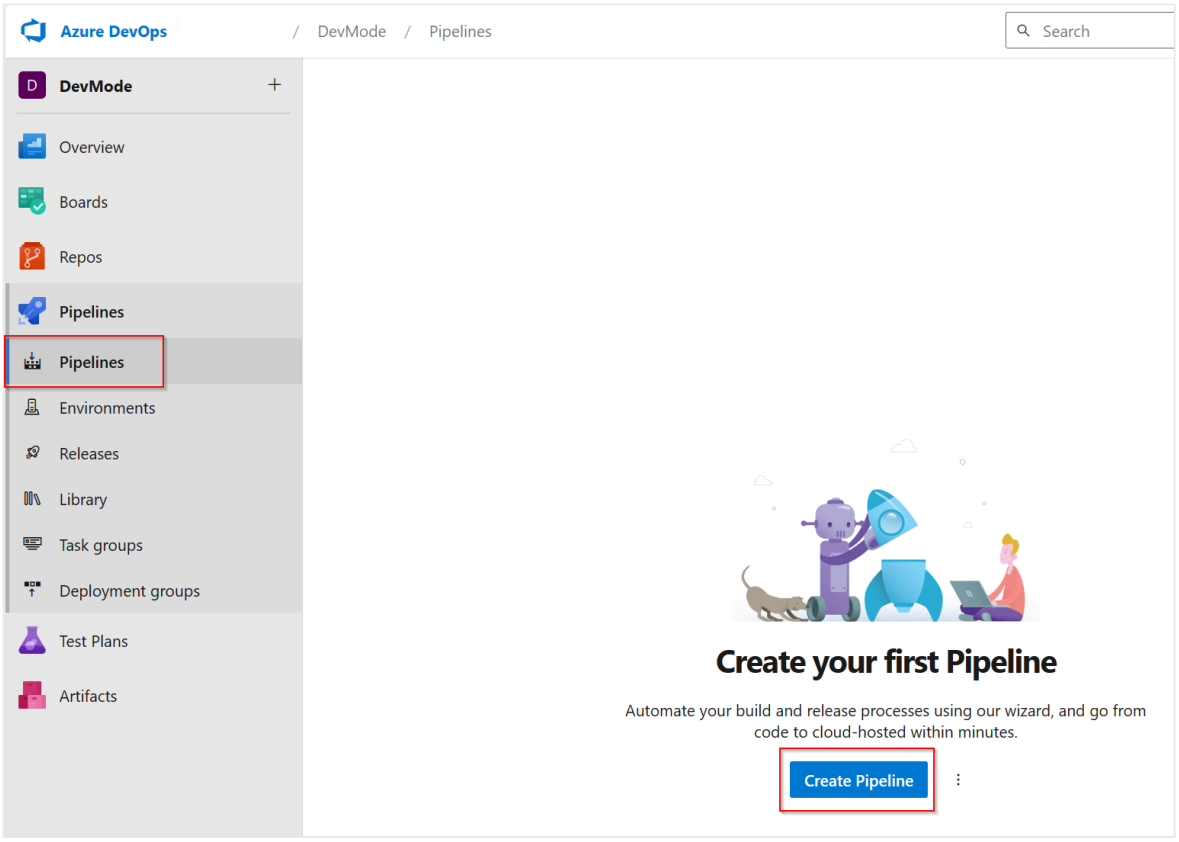
When Fabric Git integration finishes exporting your workspace items, your Azure DevOps branch contains a folder for each item in your workspace:



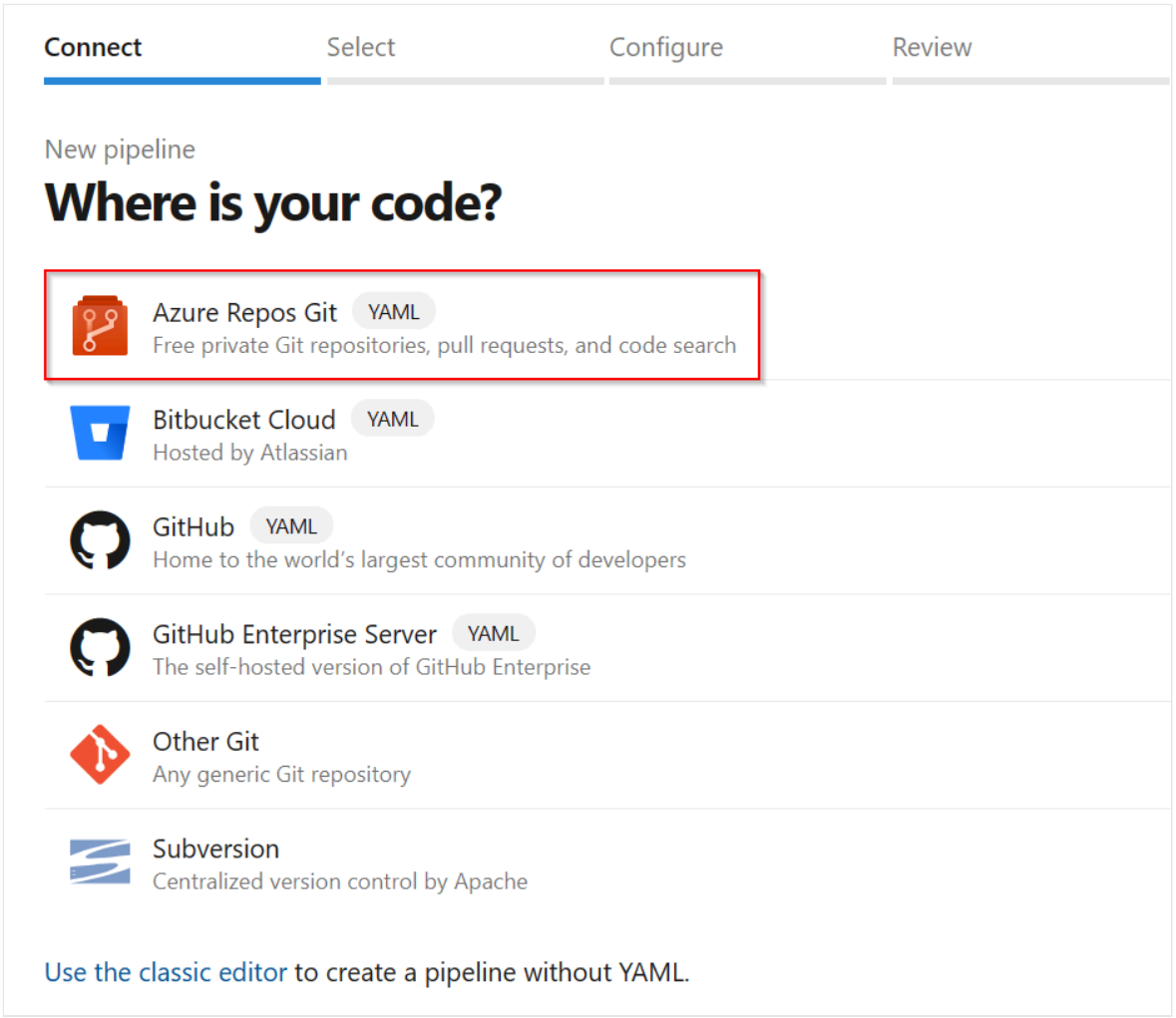
Step 2 - Create and run an Azure DevOps pipeline

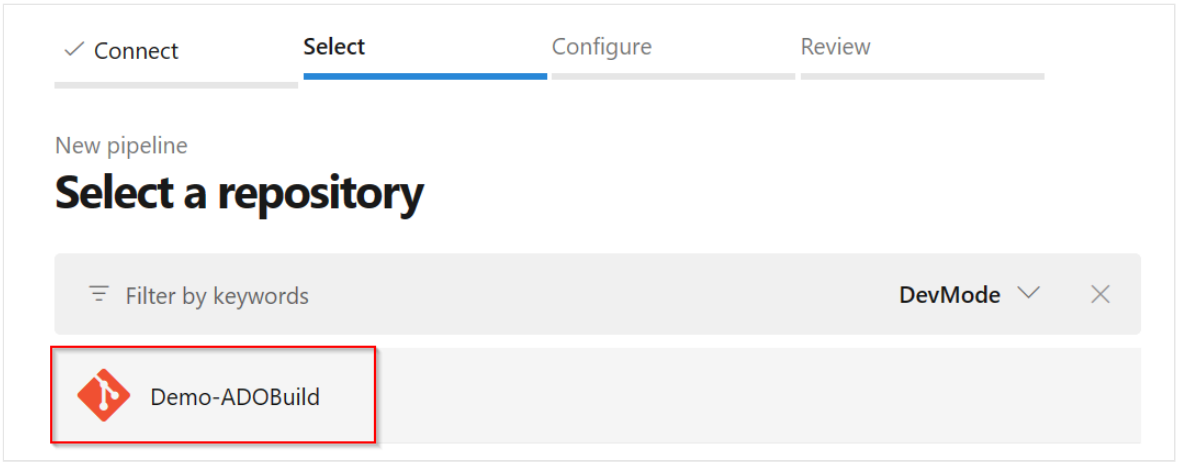
To create a new pipeline:

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

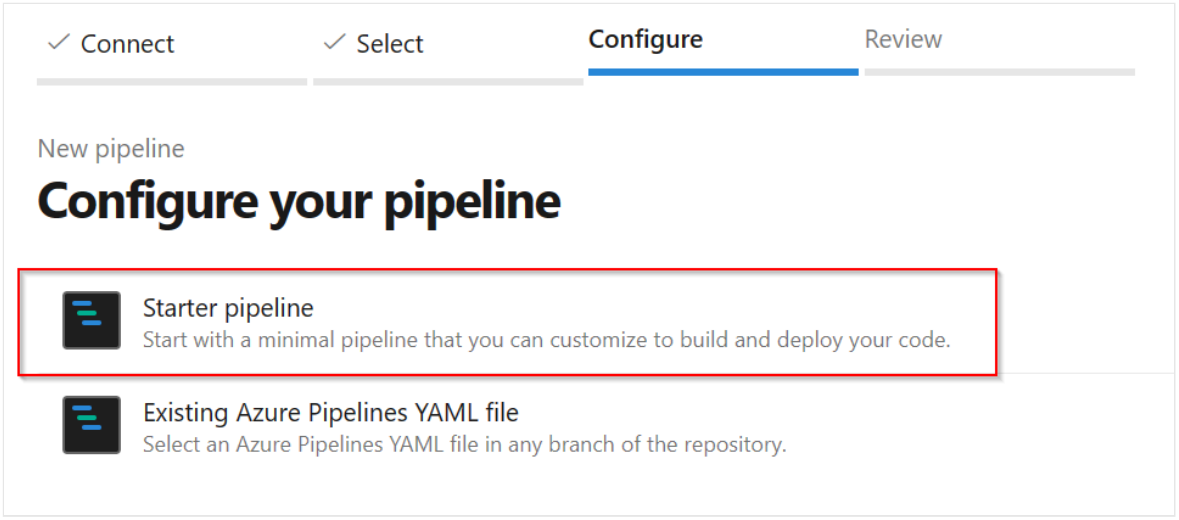


2. Select **Azure Repos Git** and select the first repository (the repository connected to the Fabric workspace):

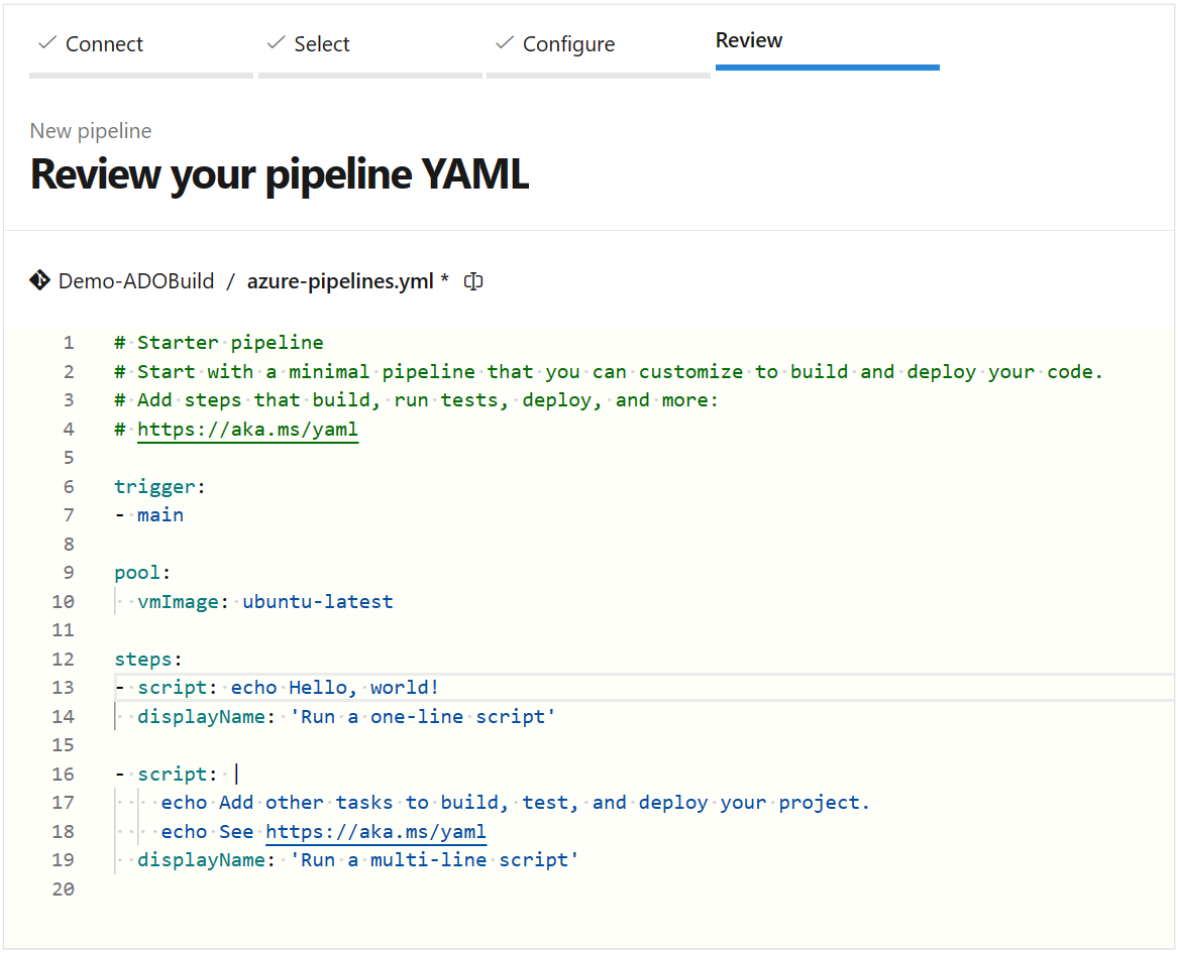




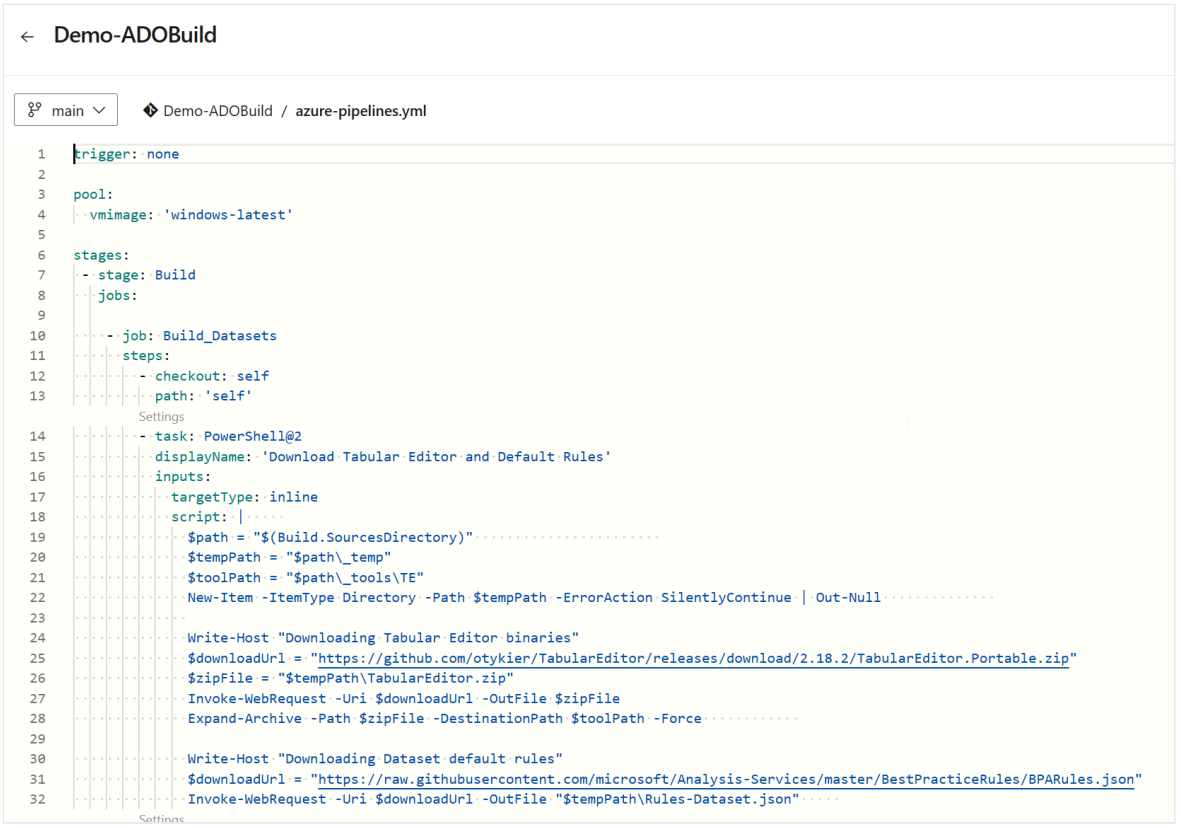
3. Select **Starter pipeline**.



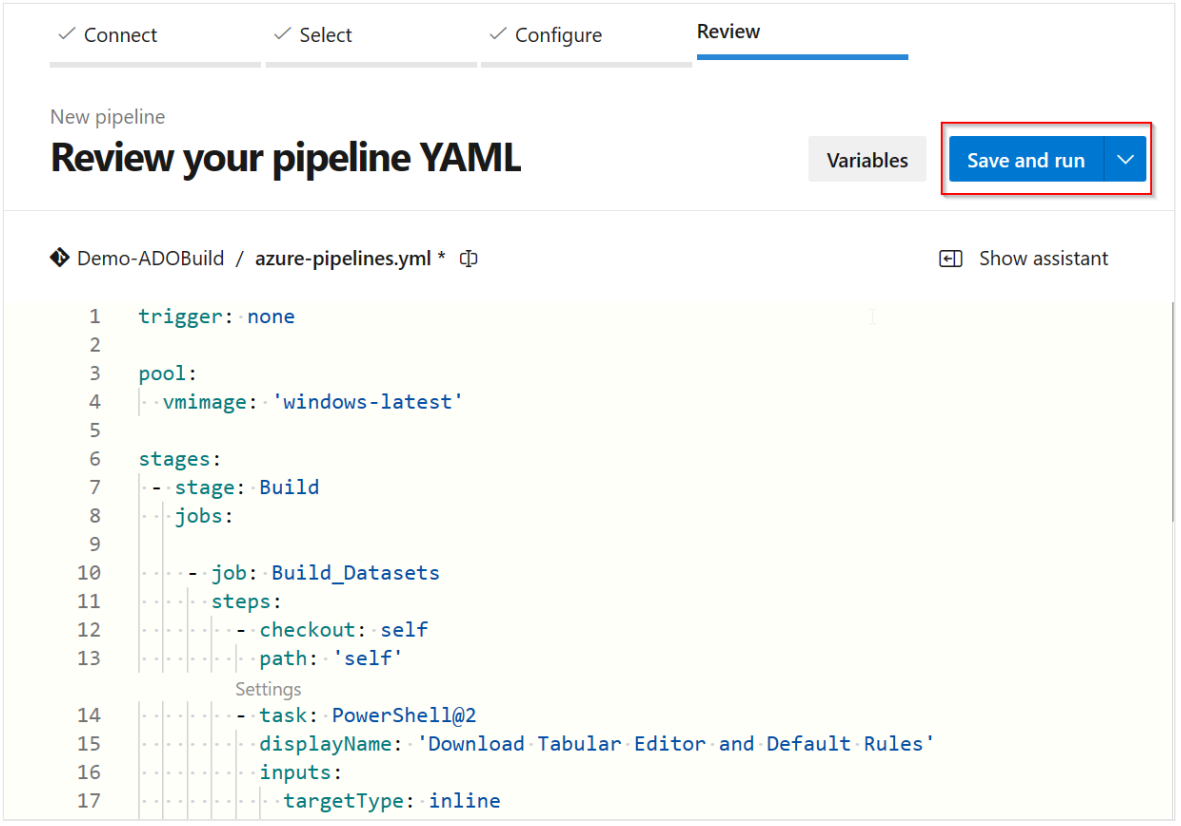
The following YAML code appears in the editor:

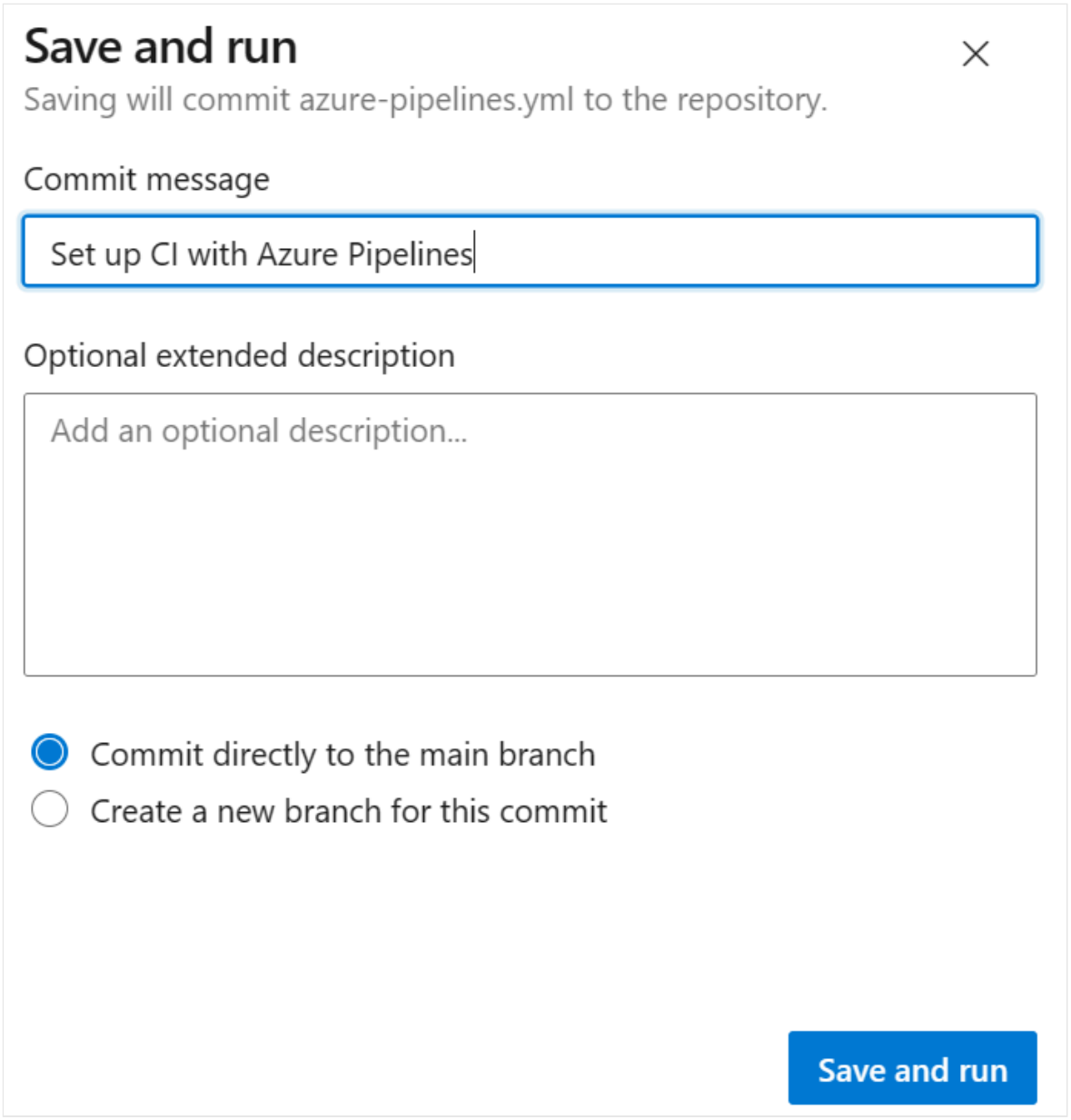


4. Copy and paste the YAML code from the Power BI developer mode pipeline into the pipeline you created:

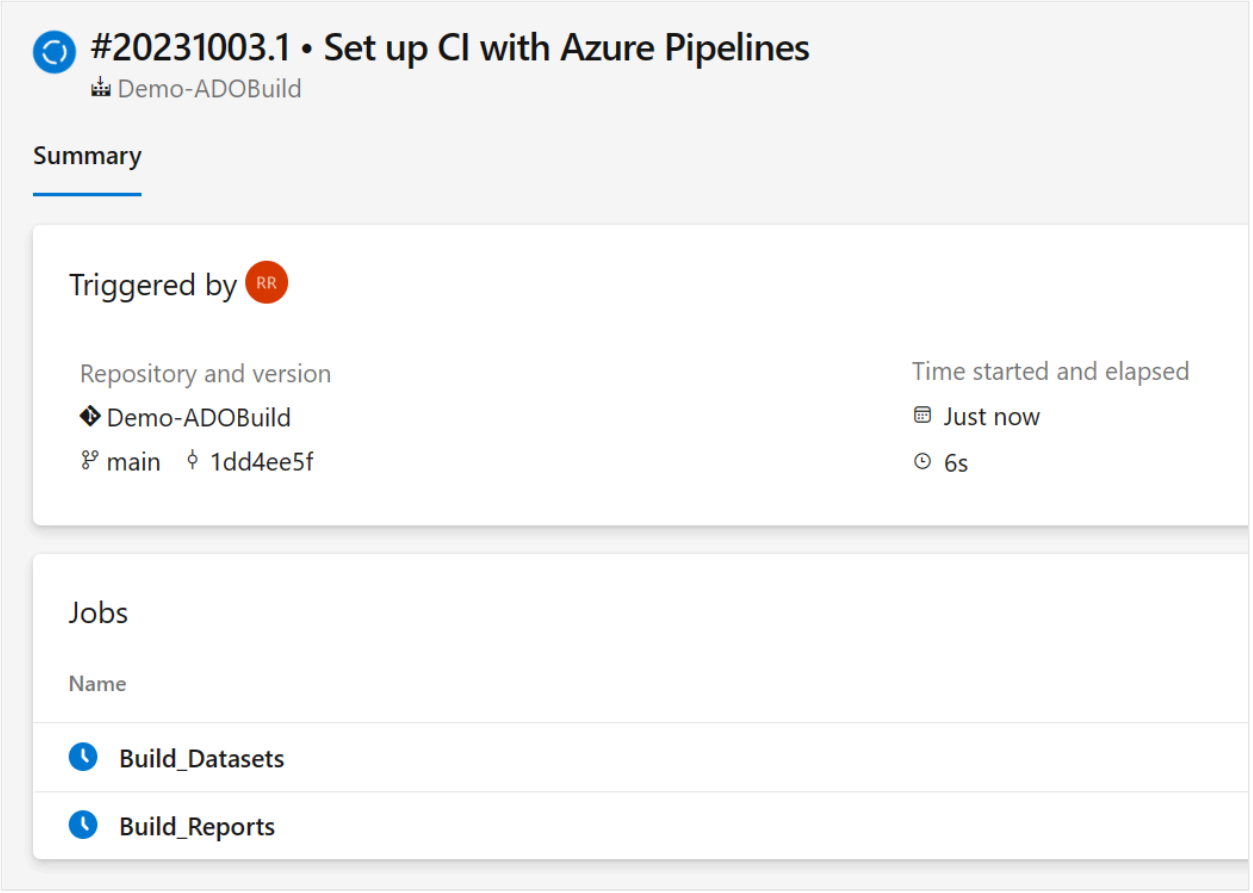


5. Select **Save and Run** to commit your new pipeline to the repository.



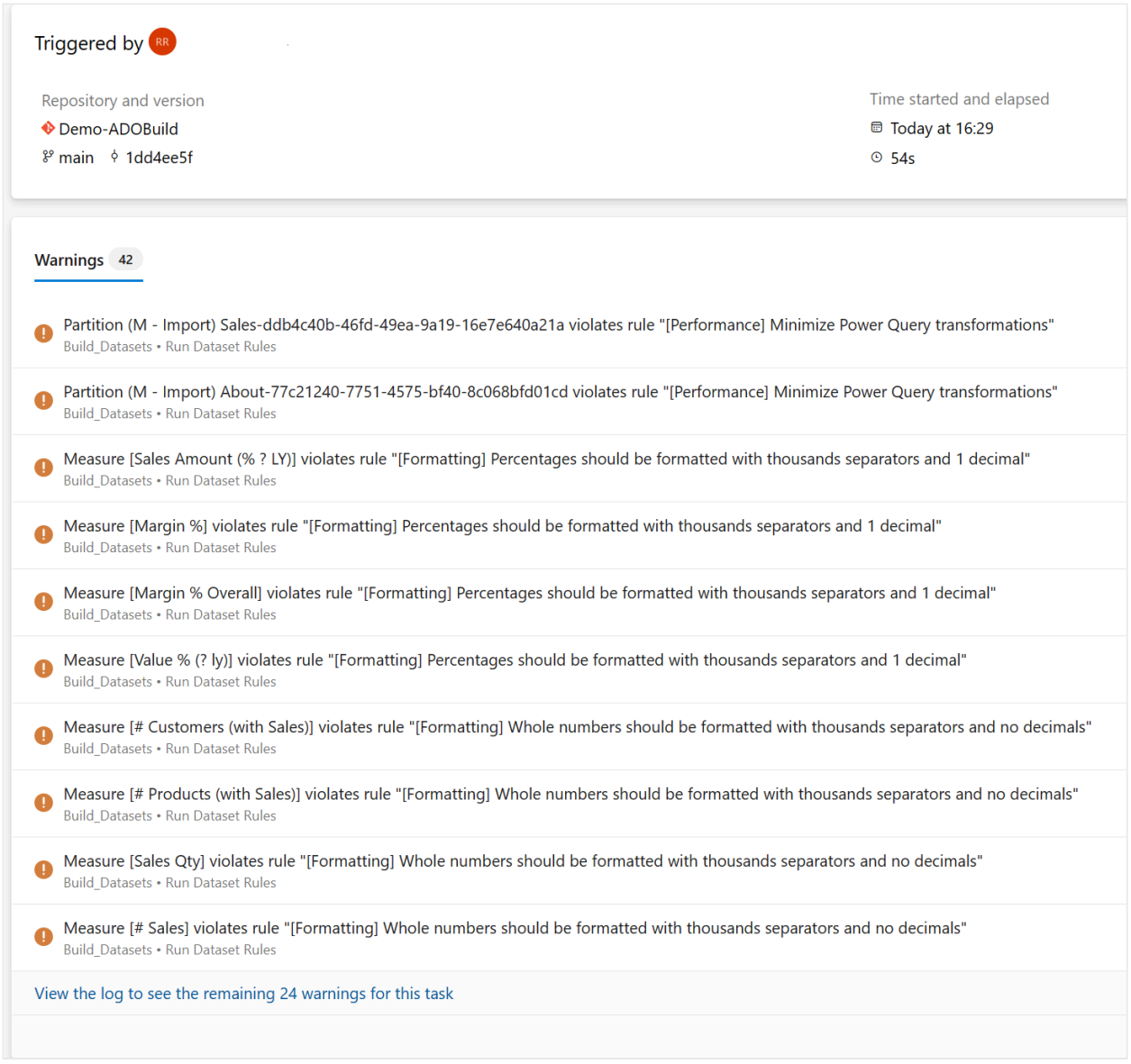


Azure DevOps runs the pipeline and starts two build jobs in parallel:

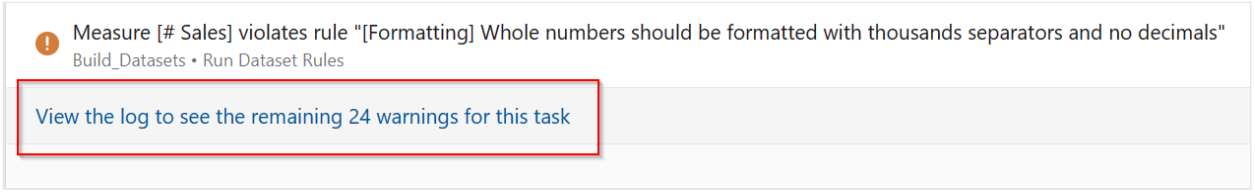


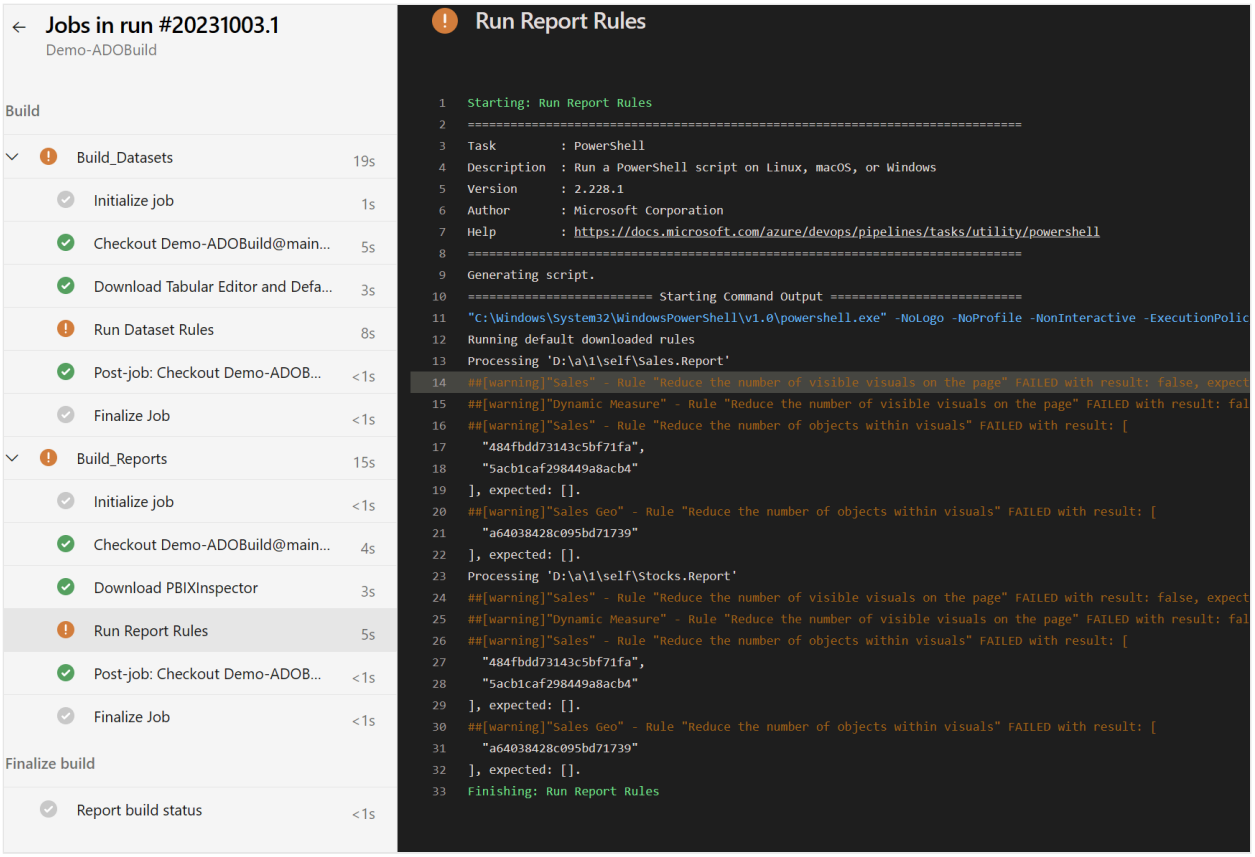
- Build_Datasets
  - Downloads Tabular Editor binaries.
  - Download Best Practice Analyzer default rules. To customize the rules, add a *Rules-Dataset.json* to the root of the repository.
  - Cycle through all the semantic model item folders and run Tabular Editor BPA Rules.
- Build_Reports
  - Download 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:

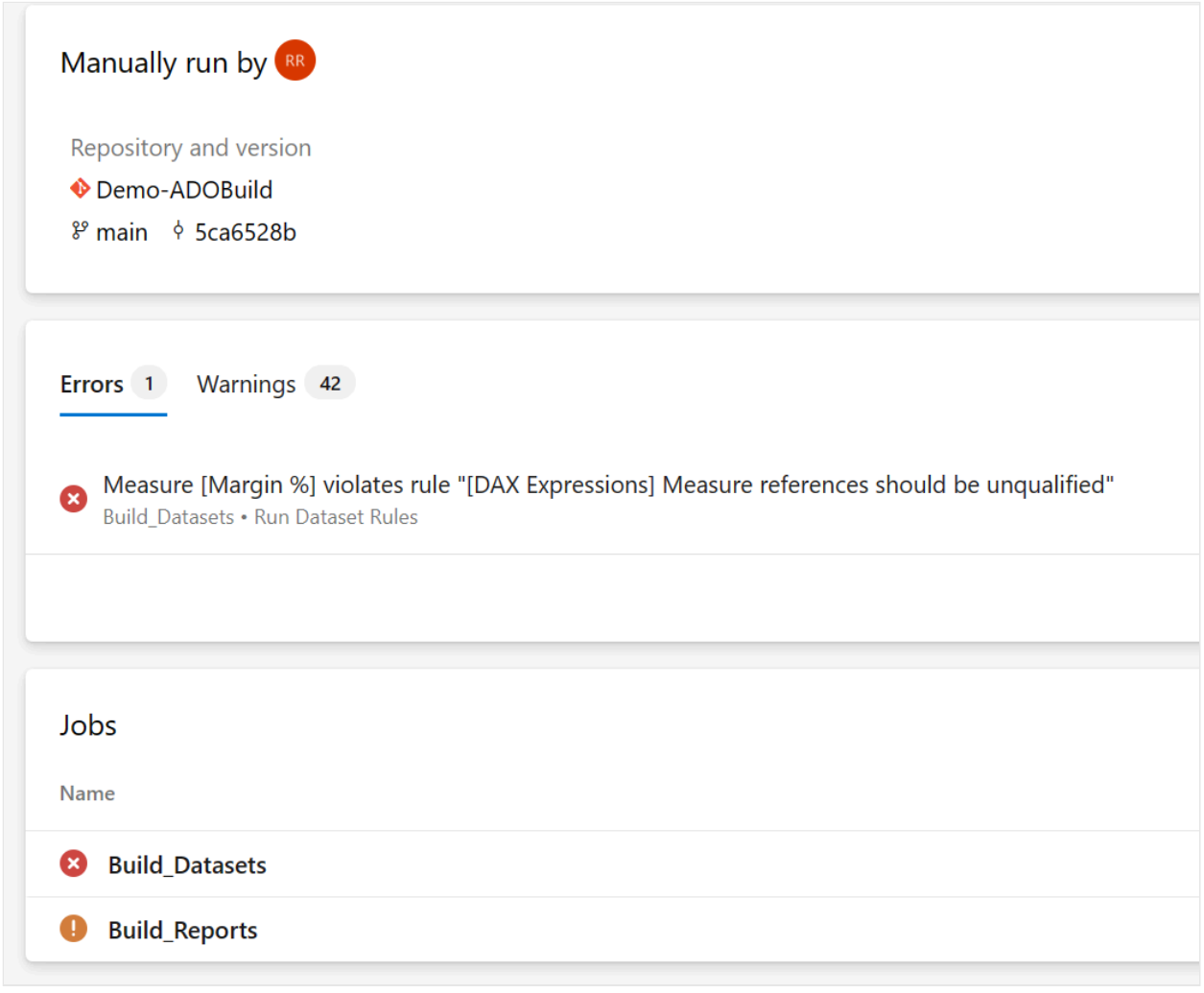


Select on the link to open a more detailed view of the two jobs:





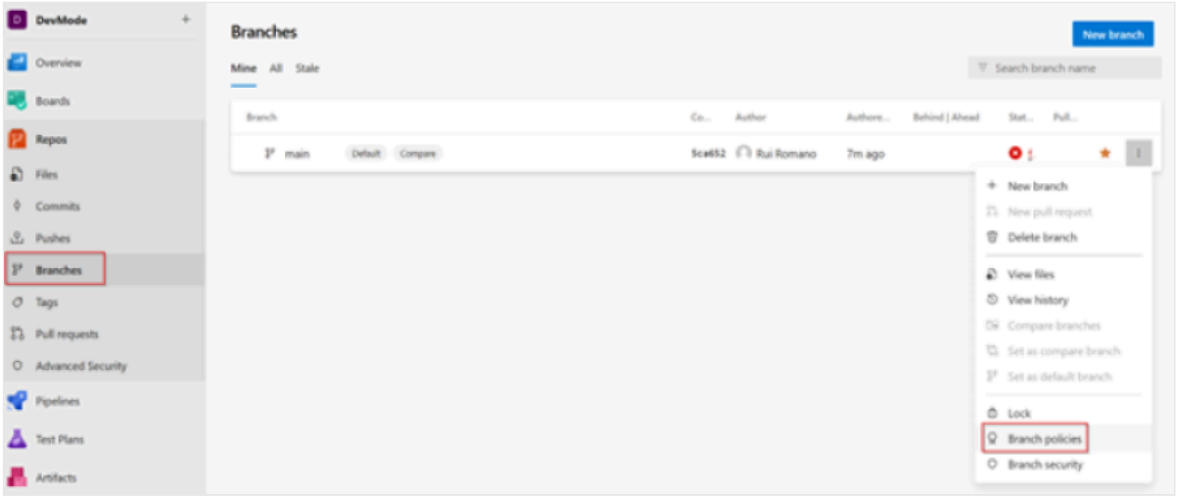
If your report or semantic model fails a rule with a higher severity level, the build fails, and the error is highlighted:



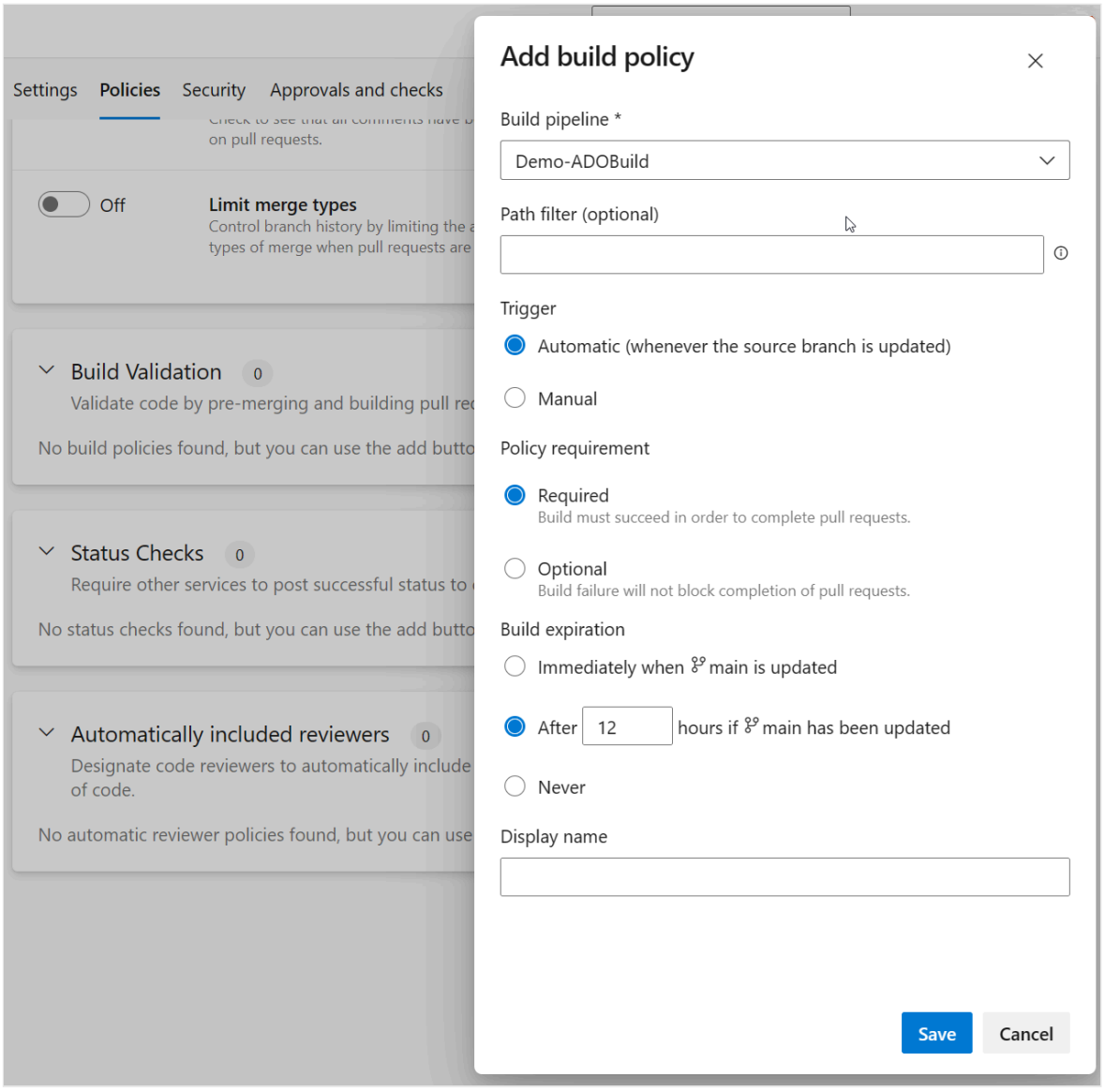
Step 3 - Define branch policies

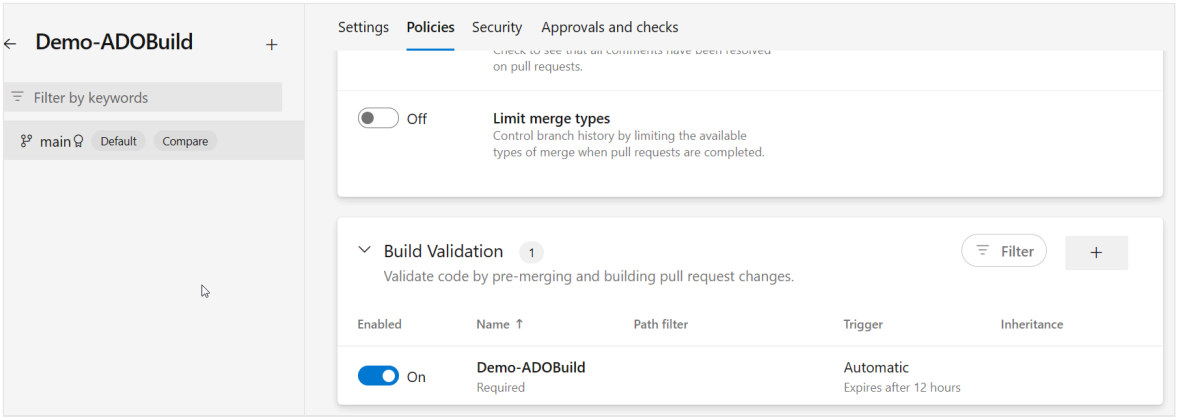
Once the pipeline is up and running, enable **Branch Policies** on the *main* branch. This step ensures that no commits can be made directly into main. A "pull request" is always required to merge changes back into *main* and you can configure the pipeline to run with every pull request.

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



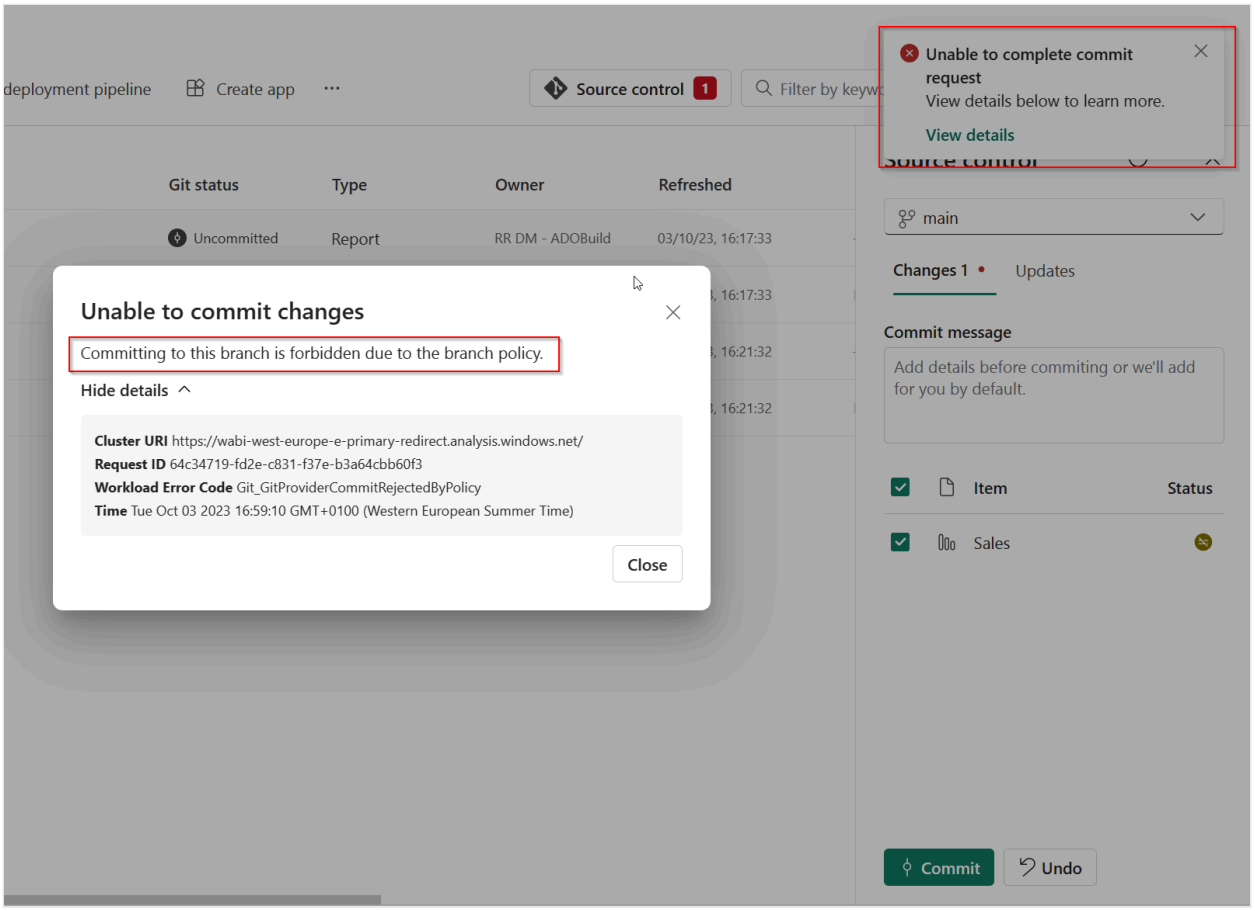
2. Configure the created pipeline as a *Build Policy* for the branch:





Step 4 - Create pull request

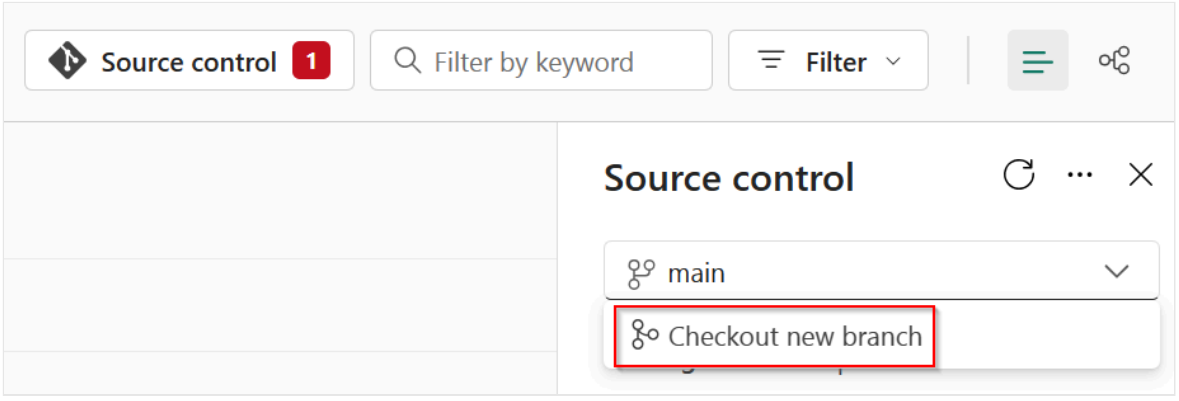
If you return to your Fabric Workspace, make a modification to one of the reports or semantic models, and attempt to commit the change, you receive the following error:

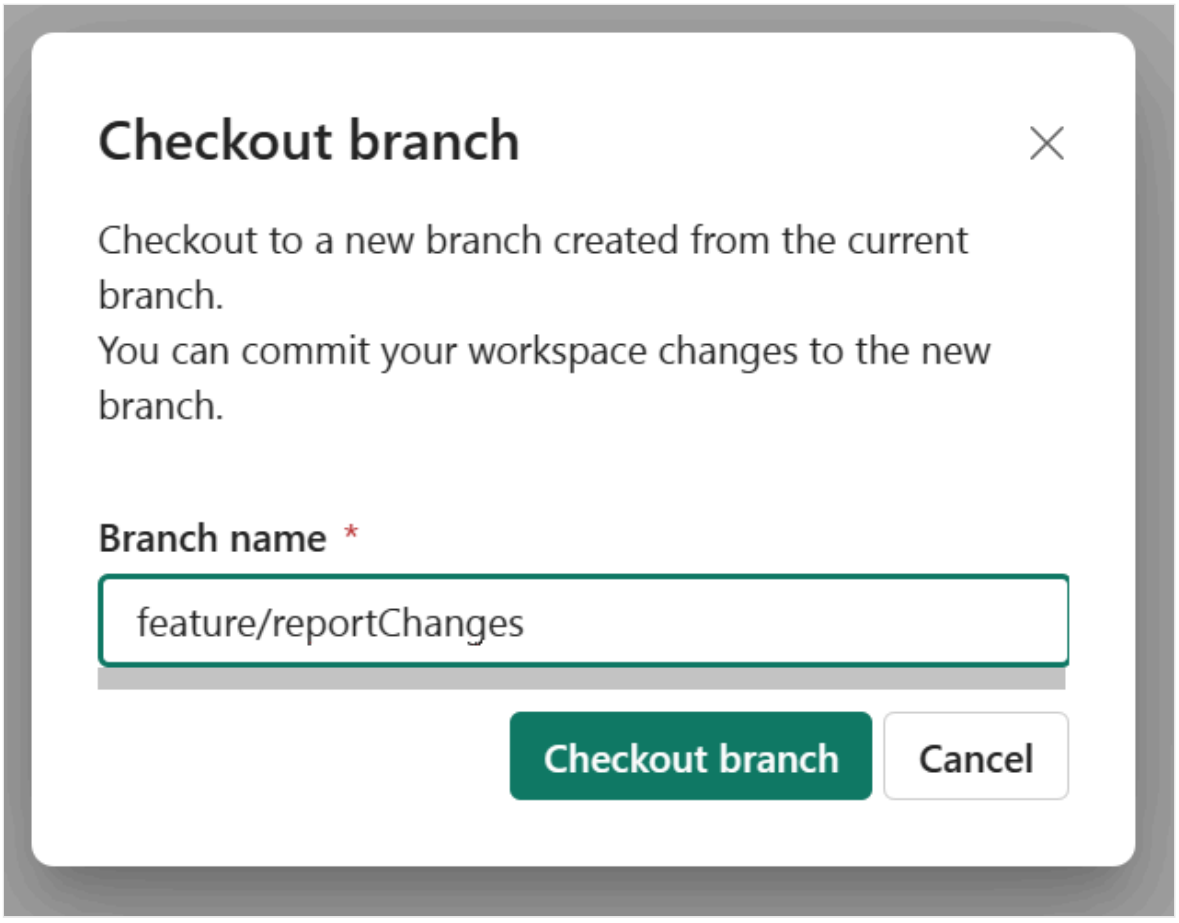


You can only make changes to the main branch through a pull request. To create a pull request checkout a new branch to make the changes on:

Create a branch directly from the Fabric Workspace:

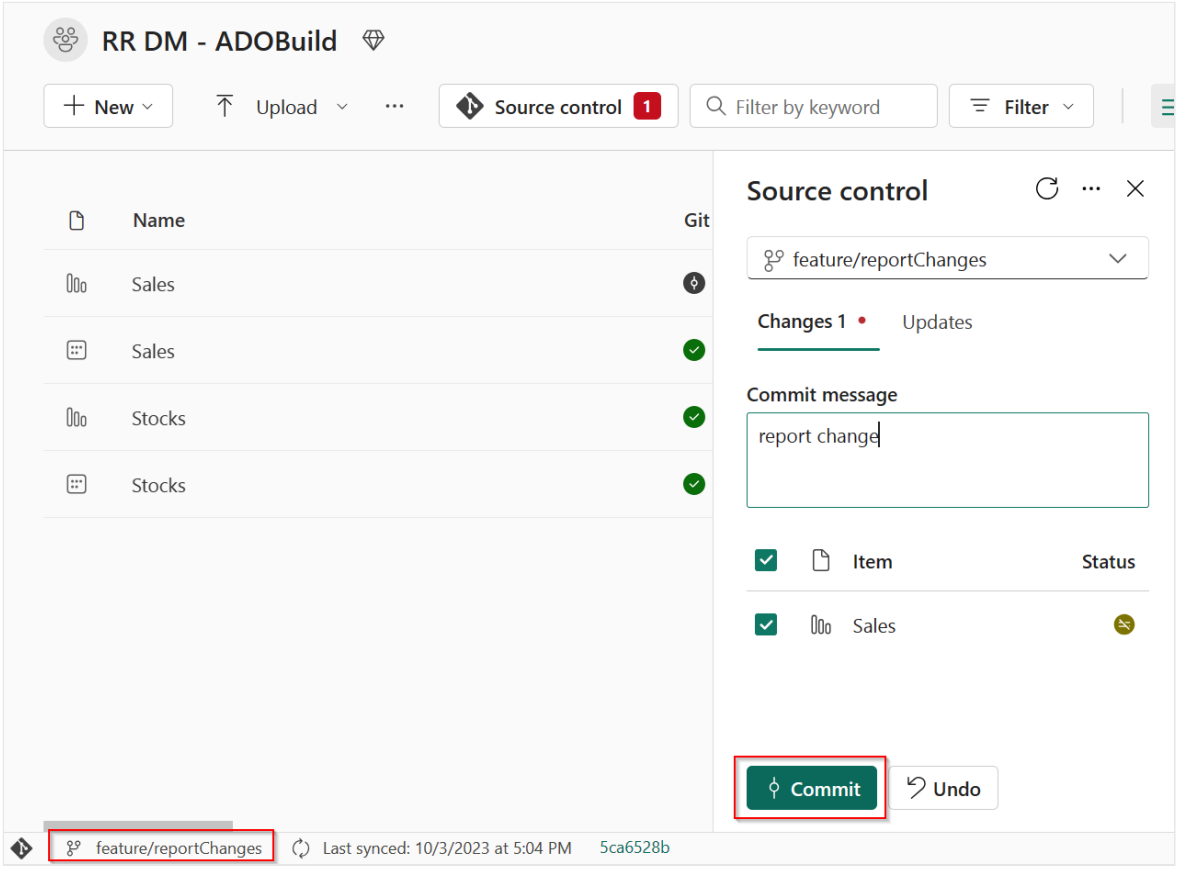
1. In the Source Control pane, select on **Checkout new branch** and provide a name for the branch.



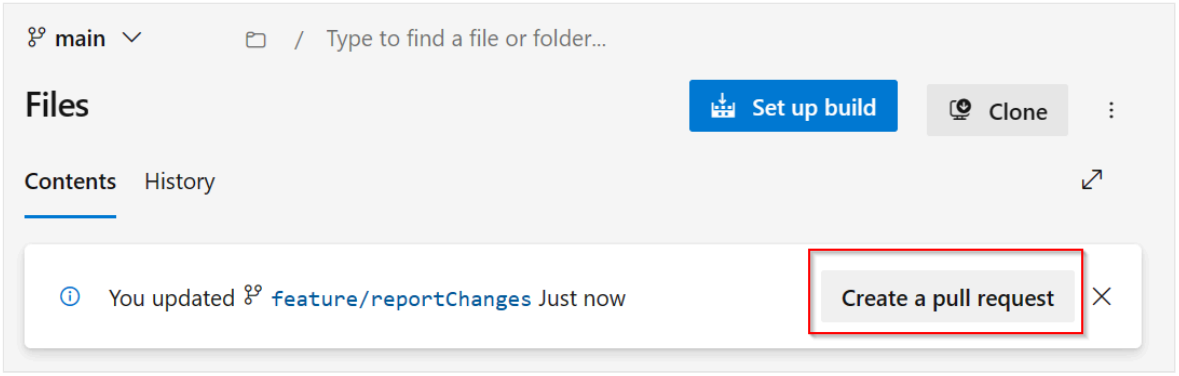


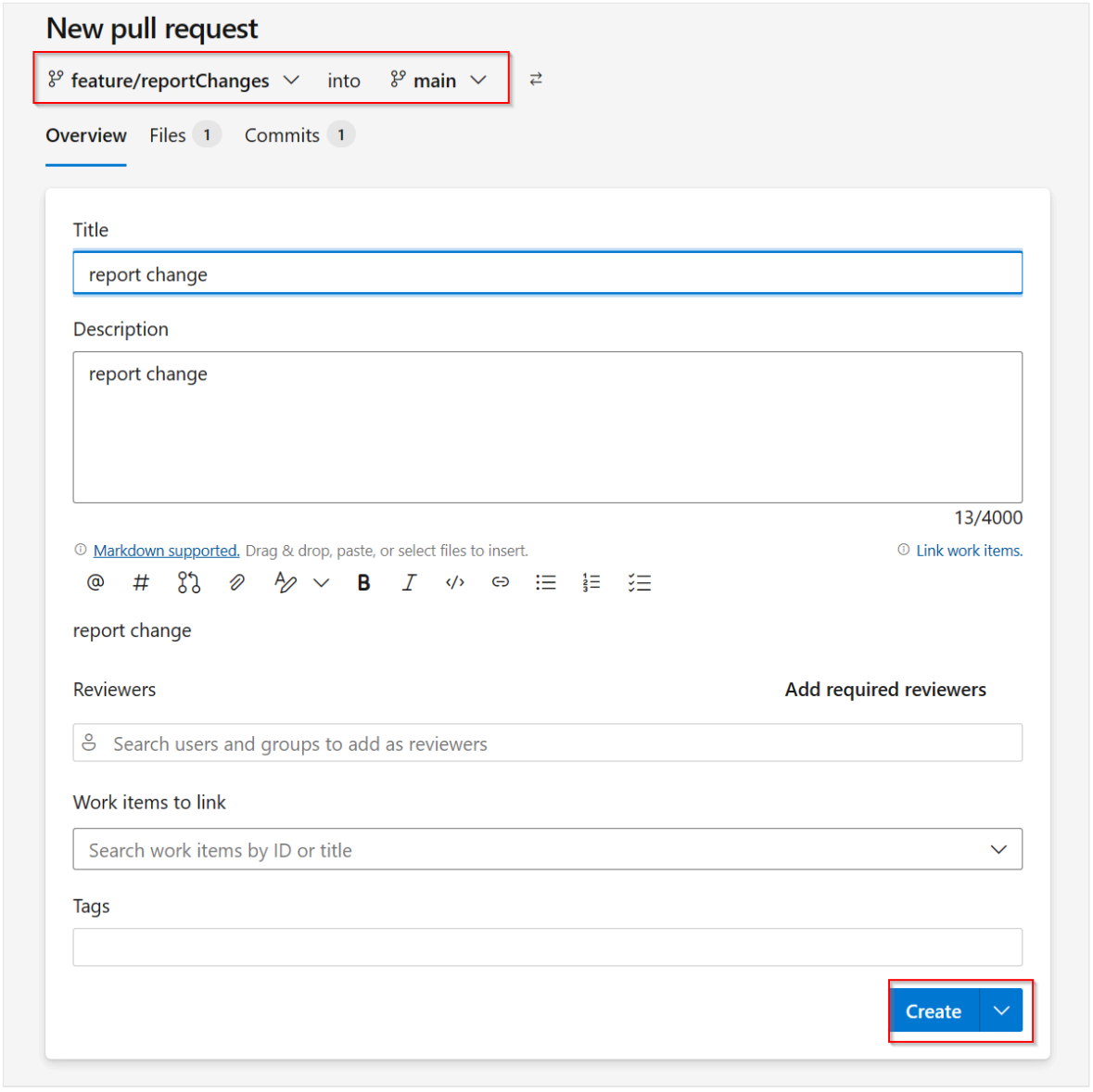
Alternatively, you can choose to develop within a separate, isolated workspace or in Power BI Desktop. For more information, see Manage Git branches

2. Commit your changes to this new branch.

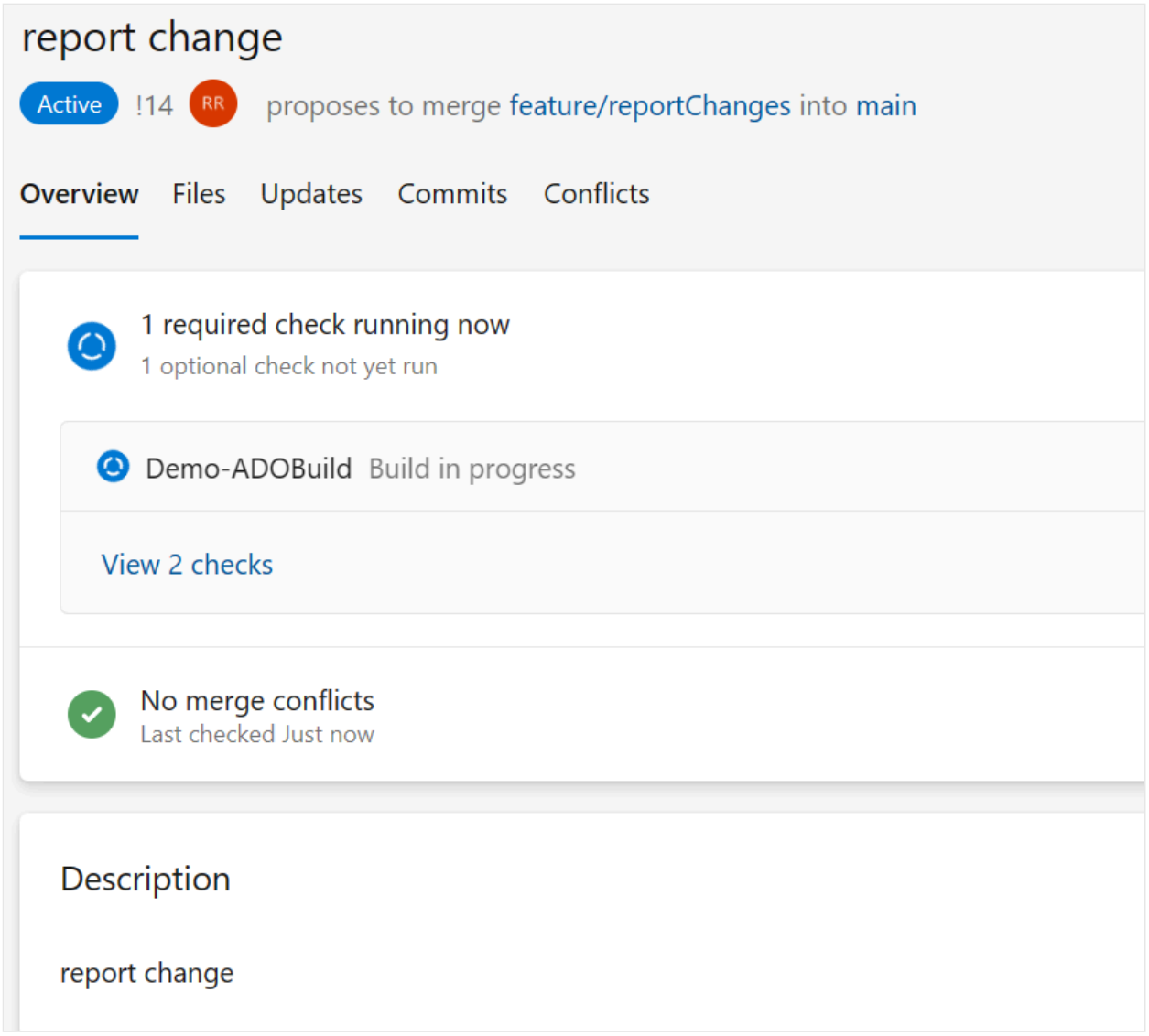


3. Following the commit, create a pull request into the *main* branch from the Azure DevOps portal.

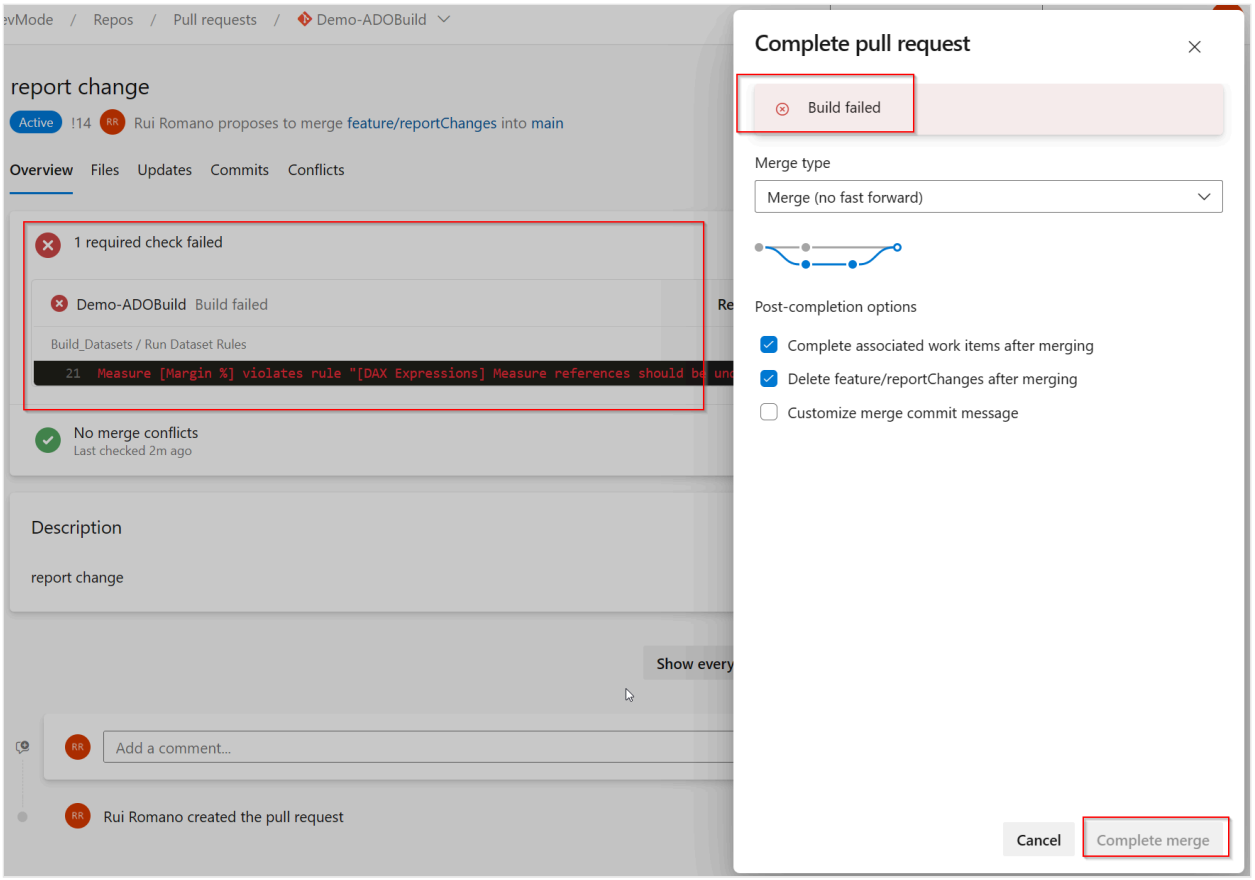




The pull request workflow not only allows you to validate and review the changes, but also automatically triggers the pipeline.



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



Related content

- For information about synchronizing your workspace with the Git branch, including updating your workspace and committing changes to Git, see Get started with Git integration.
- For tips about different options for building CI/CD processes in Fabric based on common customer scenarios, see Choose the best Fabric CI/CD workflow option for you.

Feedback

Was this page helpful? Yes No

Provide product feedback | Ask the community

Deploy a Power BI project using Fabric APIs

Article • 10/10/2024

This article explains how to deploy a Power BI project (.pbip) using Fabric REST APIs and a PowerShell script. This article is written for developers who need complete control over their Power BI deployment process and should serve as an example. The example's pattern can be applied to other languages or tools capable of calling the Fabric REST APIs.

Save your work as a Power BI project

To use the Fabric REST APIs, you need to save your work as a Power BI project file (.pbip).

Understand which APIs are used

To deploy the .pbip content, use the following Fabric REST APIs:

- List Items - Lists the existing items in the workspace.
- Create Item - Creates a new item.
- Update Item Definition - Updates the item definition. This API is used in case the item already exists.

PowerShell script

To deploy your project, use a PowerShell script. The script executes the following actions:

1. Ensures the workspace exists.
2. Creates or updates the Power BI report and semantic model in the workspace, using the Power BI Project file definitions.

The script uses the FabricPS-PBIP module, which serves as a wrapper for the Fabric APIs and handles tasks such as authentication, asynchronous calls, and metadata management for Power BI project files.

> **Note**
>
> The powershell module *fabricps-pbip* is open-source and Microsoft does not offer support or documentation for it.

Script template

Use this script template to deploy your Power BI project:

PowerShell

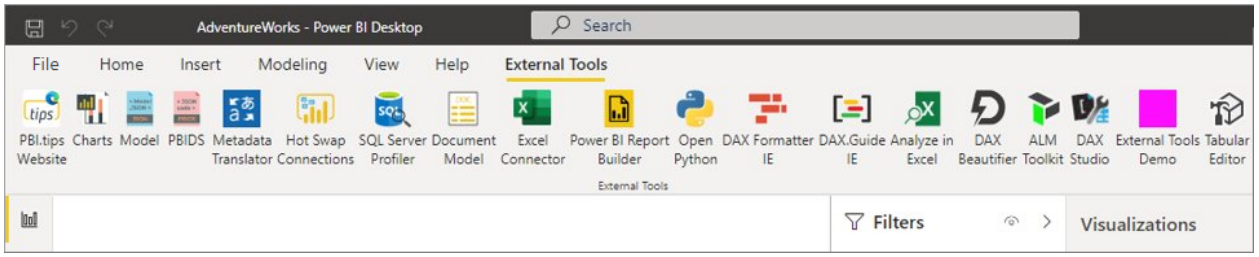
```
# Parameters
$workspaceName = "[Workspace Name]"
$pbipSemanticModelPath = "[PBIP Path]\[Item Name].SemanticModel"
$pbipReportPath = "[PBIP Path]\[Item Name].Report"
$currentPath = (Split-Path $MyInvocation.MyCommand.Definition -Parent)
Set-Location $currentPath

# Download modules and install
New-Item -ItemType Directory -Path ".\modules" -ErrorAction SilentlyContinue
| Out-Null
@("https://raw.githubusercontent.com/microsoft/Analysis-
Services/master/pbidevmode/fabricps-pbip/FabricPS-PBIP.psm1"
, "https://raw.githubusercontent.com/microsoft/Analysis-
Services/master/pbidevmode/fabricps-pbip/FabricPS-PBIP.psd1") |% {
    Invoke-WebRequest -Uri $_ -OutFile ".\modules\$(Split-Path $_ -Leaf)"
}
if(-not (Get-Module Az.Accounts -ListAvailable)) {
    Install-Module Az.Accounts -Scope CurrentUser -Force
}
Import-Module ".\modules\FabricPS-PBIP" -Force

# Authenticate
Set-FabricAuthToken -reset

# Ensure workspace exists
$workspaceId = New-FabricWorkspace  -name $workspaceName -skipErrorIfExists

# Import the semantic model and save the item id
$semanticModelImport = Import-FabricItem -workspaceId $workspaceId -path
$pbipSemanticModelPath

# Import the report and ensure its binded to the previous imported report
$reportImport = Import-FabricItem -workspaceId $workspaceId -path
$pbipReportPath -itemProperties @{"semanticModelId" =
$semanticModelImport.Id}
```

External tools in Power BI Desktop

Article • 09/04/2024

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 tool categories

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 *.pbitool.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. Except for the underlying public Microsoft APIs, Microsoft doesn't provide support or documentation for external tools. Microsoft does provide support if the issue can be reproduced with Microsoft tools. These tools include SQL Server Management Studio (SSMS), or sample code that uses the public Microsoft APIs.

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:

Expand table

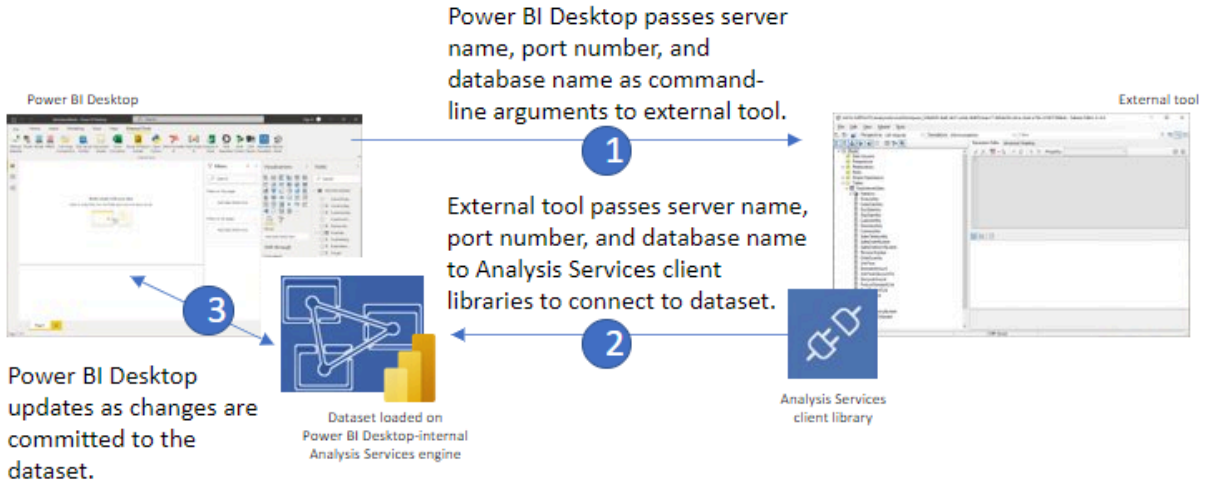
Tool	Description
PowerBI.tips - Business Ops	An easy to use deployment tool for adding external tools extensions to Power BI Desktop. The Business Ops goal is to provide a one stop shop for installing all the latest versions of external tools. To learn more, go to PowerBI.tips - Business Ops.
Tabular 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,

Tool	Description
	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:

Expand table

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 isn't 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 *.pbitool.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 *.pbitool.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.

Related content

- Register an external tool

Feedback

Register an external tool

Article • 01/15/2024

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 pbitool.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>.pbitool.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 **.pbitool.json** extension are loaded by Power BI Desktop upon startup.

Example

The following *.pbitool.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.

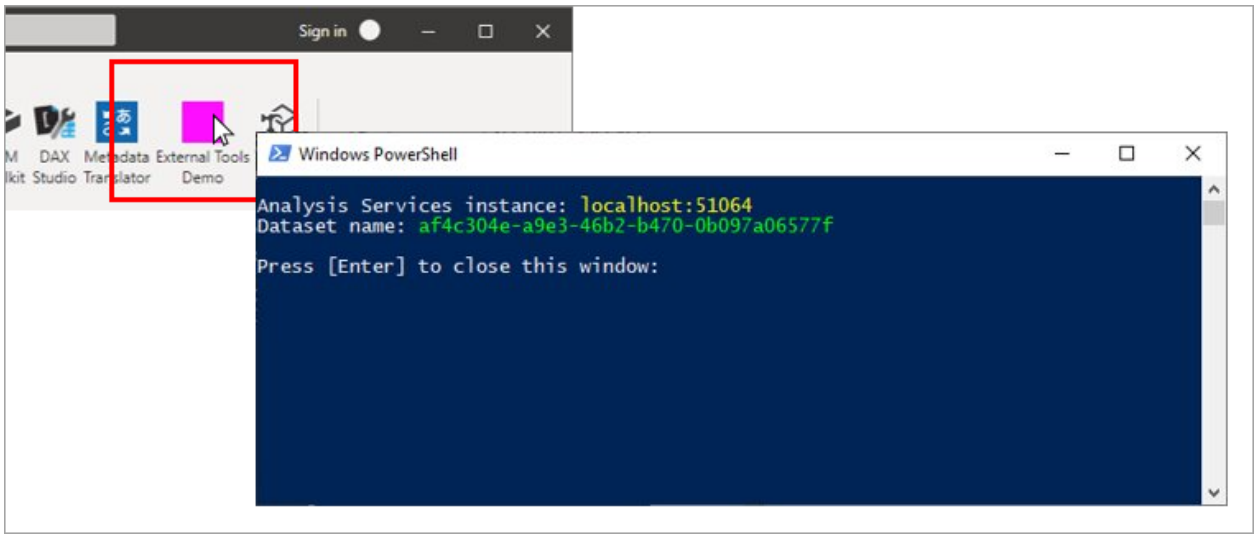
JSON

```
{
    "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,iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAYAAAAfFcSJAAAAAXNSR0IArs
4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsEAAA7BAbiRa+0AAAANSURBVBhXY/jH9+8/AA
ciAwpql7QkAAAAAElFTkSuQmCC"
}
```

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

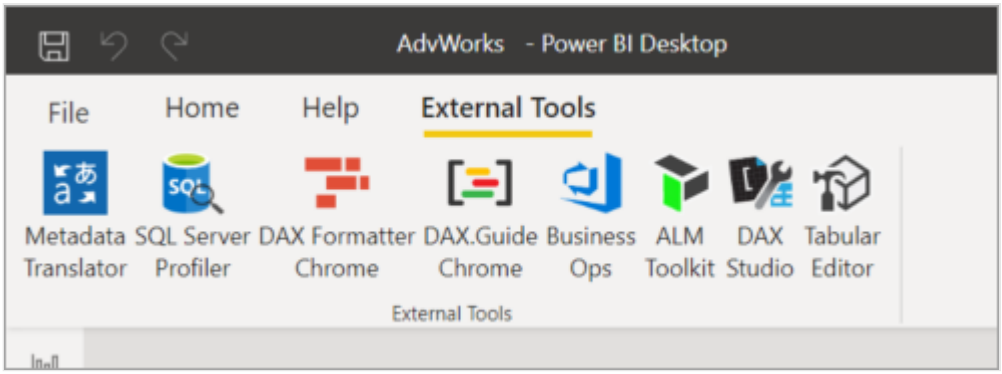
PowerShell

```
[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 pbitool.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:

``

Be sure to remove the **data:** prefix, as shown in the pbitool.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();
}
```

Related content

- External tools in Power BI Desktop
- Analysis Services client libraries
- Tabular Object Model (TOM)