#Milano

Microsoft Fabric lifecycle management in enterprise scenarios

Paolo Ranzani

Lead Data&AI Architect @ Avvale

#GLOBALAZURE 2026

#Milano

improove

TD SYNNEX

Grazie ai nostri sponsor 🙏

Who I am

5+ years of experience designing, implementing, and managing scalable cloud data architectures, mainly based on Microsoft Azure, Microsoft Fabric, Databricks, Hashicorp Terraform, Azure DevOps, and GitHub

Paolo Ranzani

Lead Data&AI Architect @Avvale

Associate @Sloweb

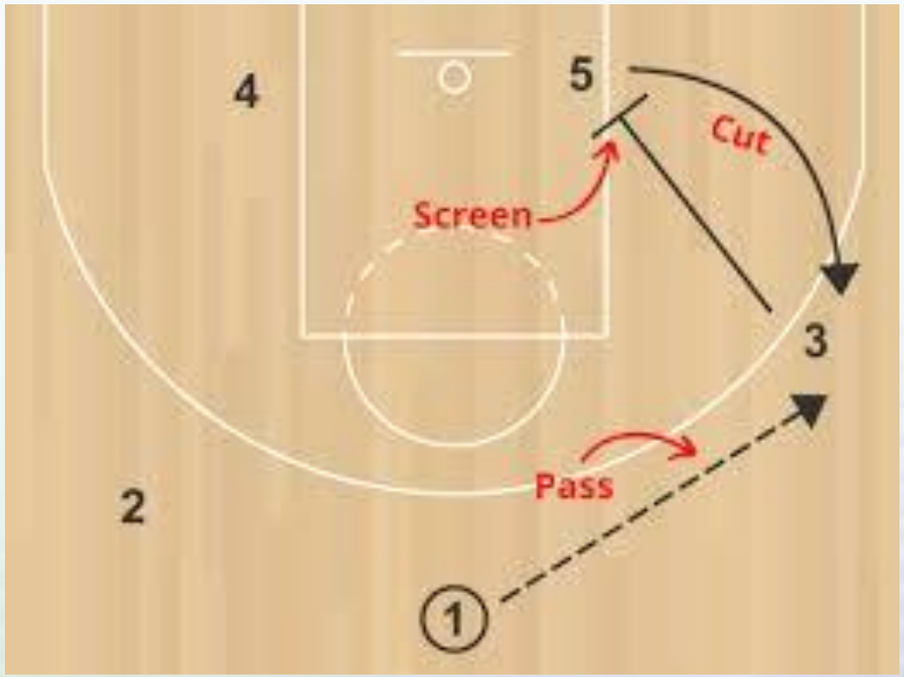
Agenda

- Introduction
- Scenario #1: Baseline Git collaboration for Data Engineers
- Scenario #2: Orchestrate Fabric Deployment pipelines with Azure DevOps
- Scenario #3: Azure DevOps Pipelines with the “fabric-cicd” Python Library
- Scenario #4: Baseline Git collaboration for PBI Developers
- Scenario #5: Power BI quality checks in Azure DevOps using BPA or PBI Inspector tools
- Scenario #6: Overview of 3rd party integrations (SonarQube)

Motivations

Speed: Players enjoy “run in transition”, fast-break points that end with big dunks

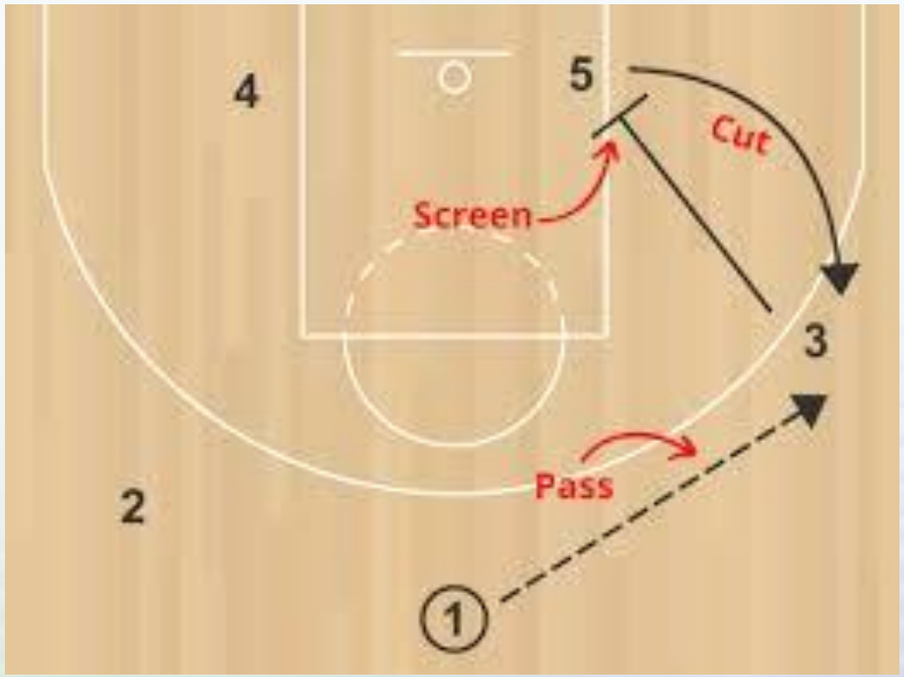
Control: Coaches ask to slow down and “execute” in the salient moment



Motivations

Speed: Players enjoy “run in transition”, fast-break points that end with big dunks

Control: Coaches ask to slow down and “execute” in the salient moment



Find the right balance to manage the pace of the game

Motivations

Speed

- Business demands delivery of BI solutions as quickly as possible
- Developers demand the freedom to release new features into production instantly

Control

- System Admins must ensure stability, compliance and security on each release
- Enterprise Customers require adhering to predefined policies, or integrating with existing automation processes

improove

TD SYNNEX

Motivations

Speed

- Business demands delivery of BI solutions as quickly as possible
- Developers demand the freedom to release new features into production instantly

Control

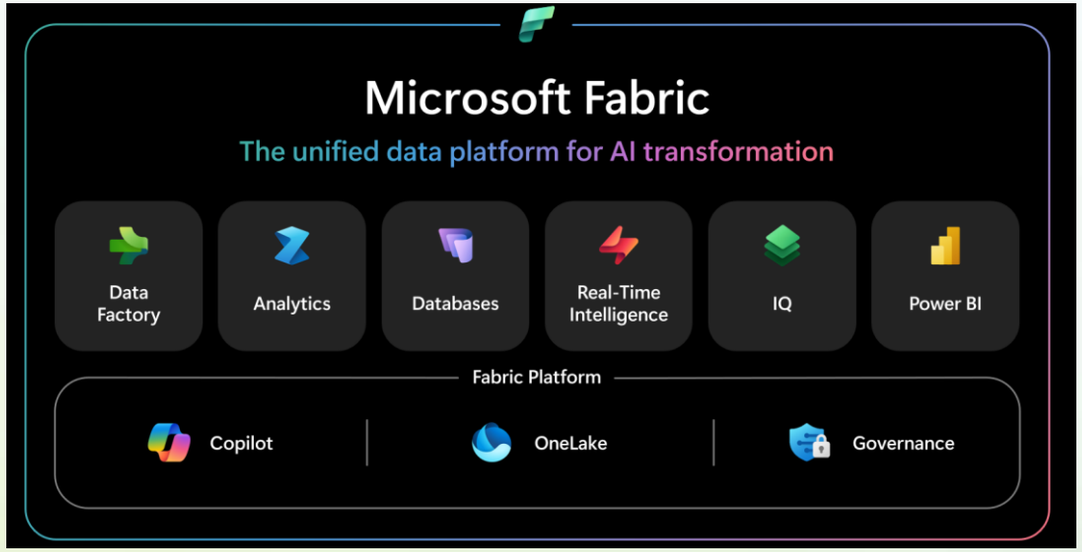
- System Admins must ensure stability, compliance and security on each release
- Enterprise Customers require adhering to predefined policies, or integrating with existing automation processes

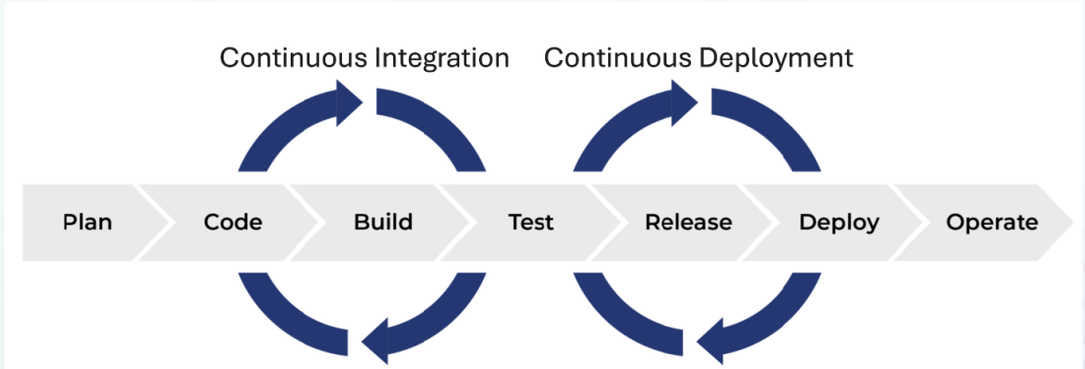
DevOps practices

improove

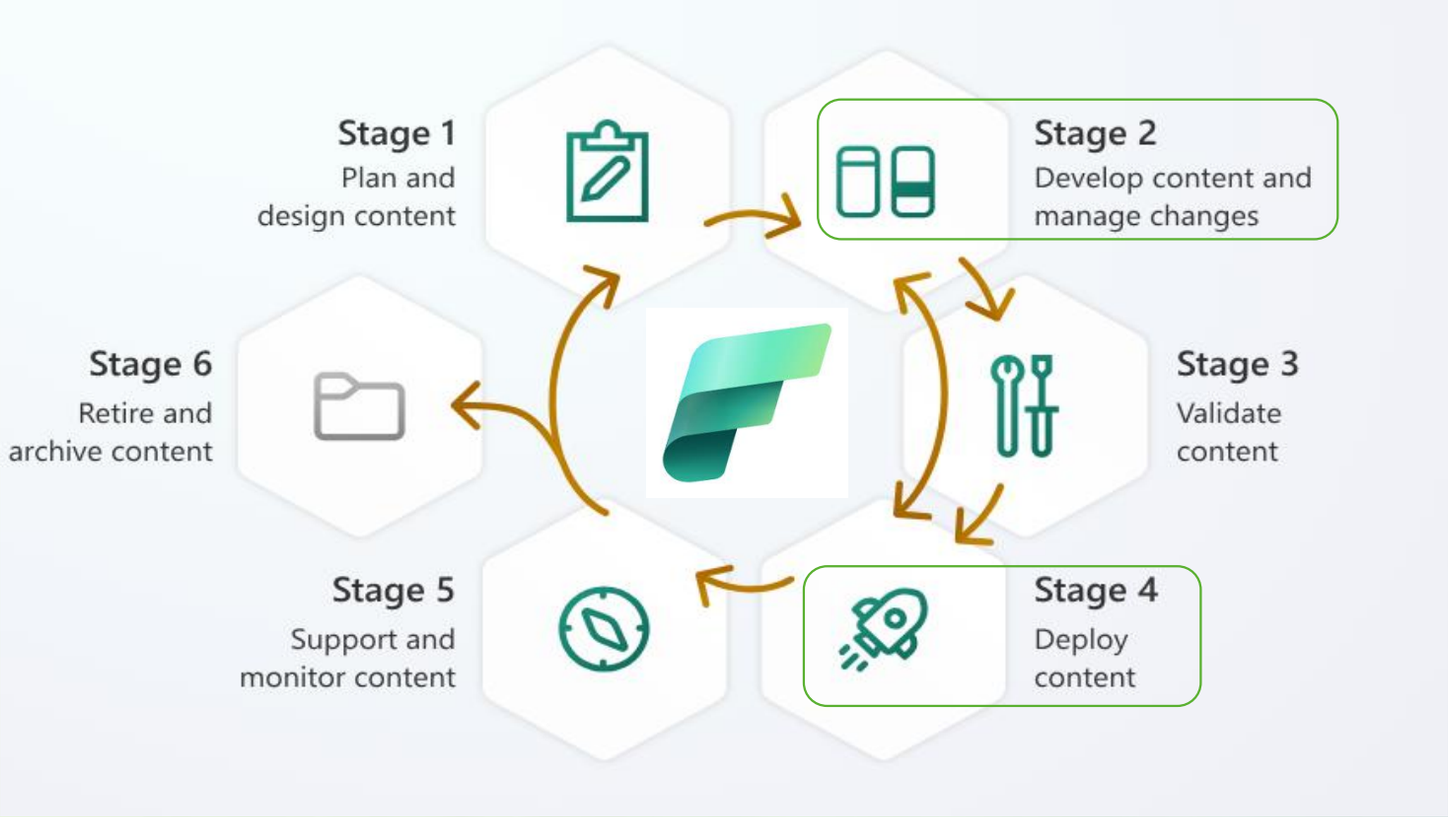
TD SYNNEX

Goal: how to merry Microsoft Fabric and DevOps practices



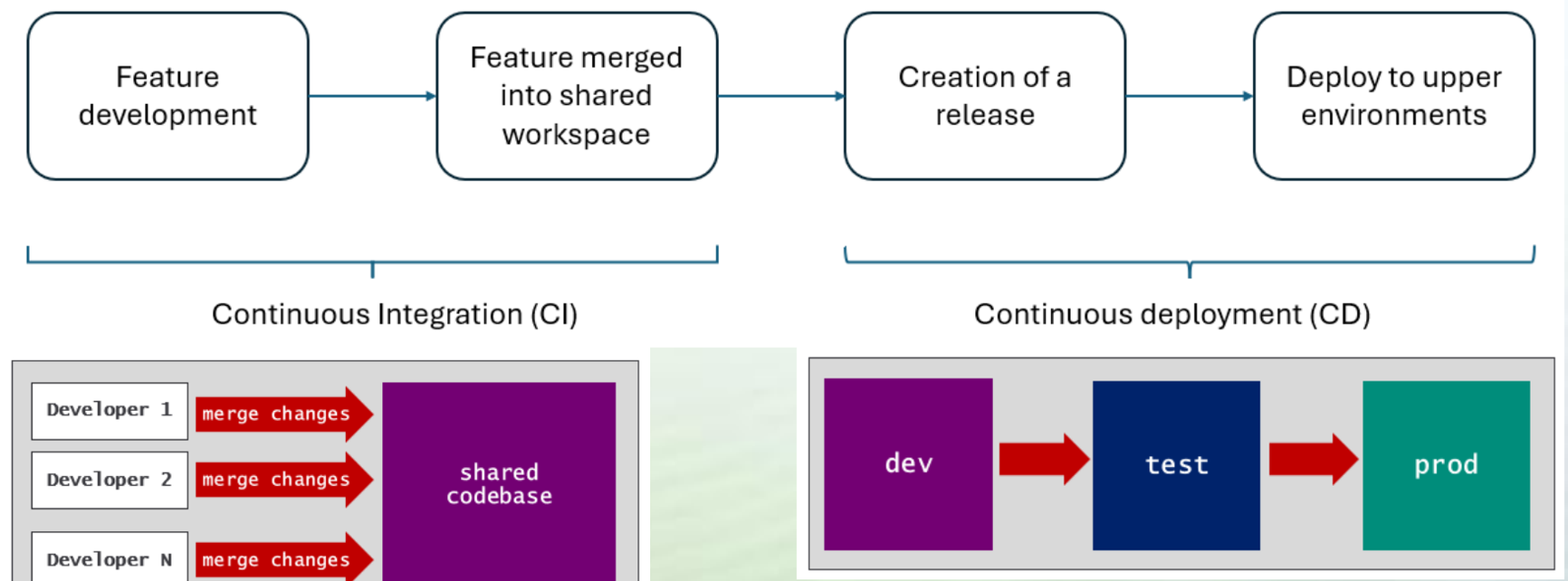


Goal: how to merry Microsoft Fabric and DevOps practices



What does CI/CD mean in MS Fabric?

- **Continuous integration (CI):** the practice of frequently merging code changes to shared Git repository
- **Continuous deployment (CD):** the practice of automating deployment of items/artifacts to target environments



Keywords:

- CI/CD
- DevOps
- Continuous Delivery
- Application Lifecycle Management (ALM)
- Platform Engineering

improove

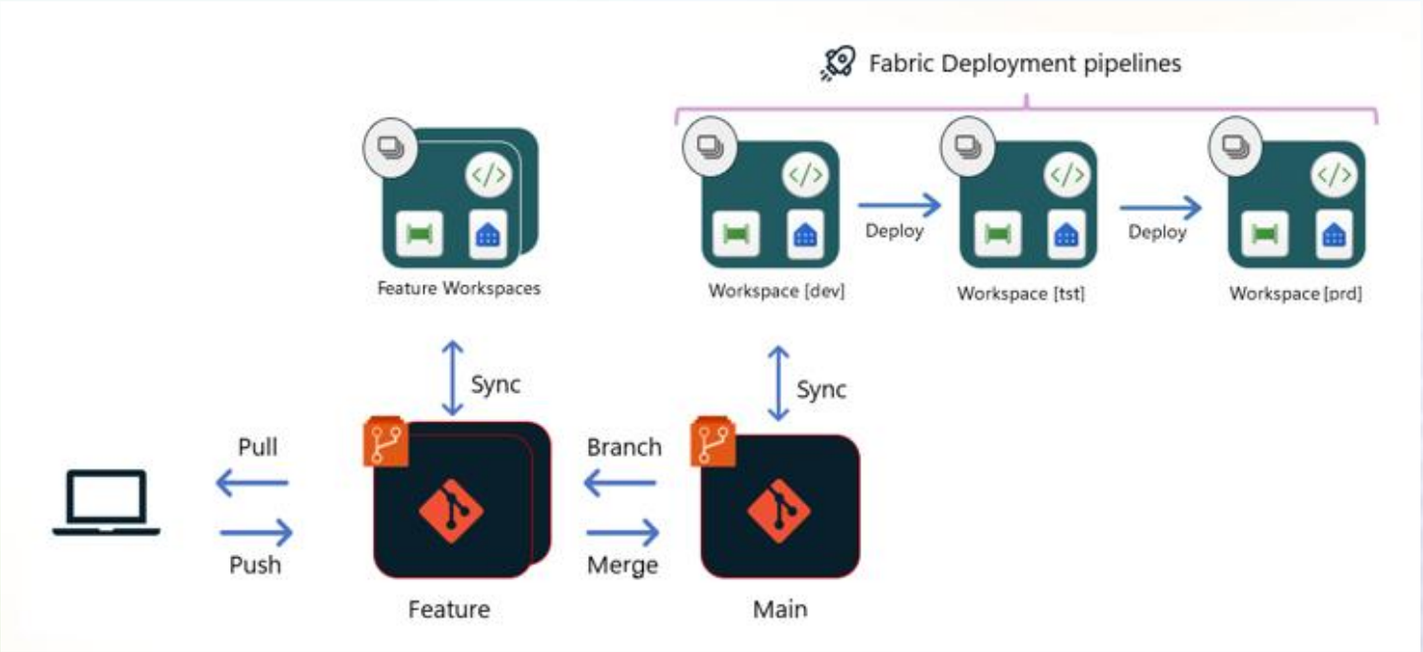
TD SYNNEX

Scenario #1: Git collaboration for Data Engineers

Use case: end-to-end **Medallion architecture** (Bronze, Silver, and Gold layers) to process raw data into actionable business insights

- **Data engineers** developing data integration and/or data processing workflows through Fabric Data Pipelines and Fabric Notebooks

- **CI:** Git Integration & Feature workspaces
- **CD:** Fabric Deployment Pipelines

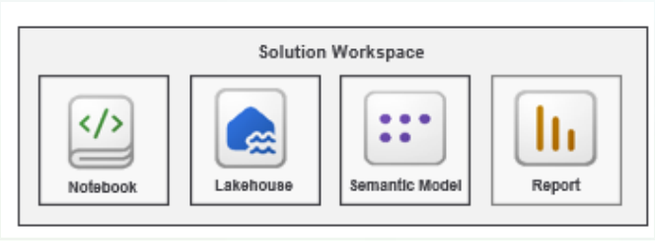


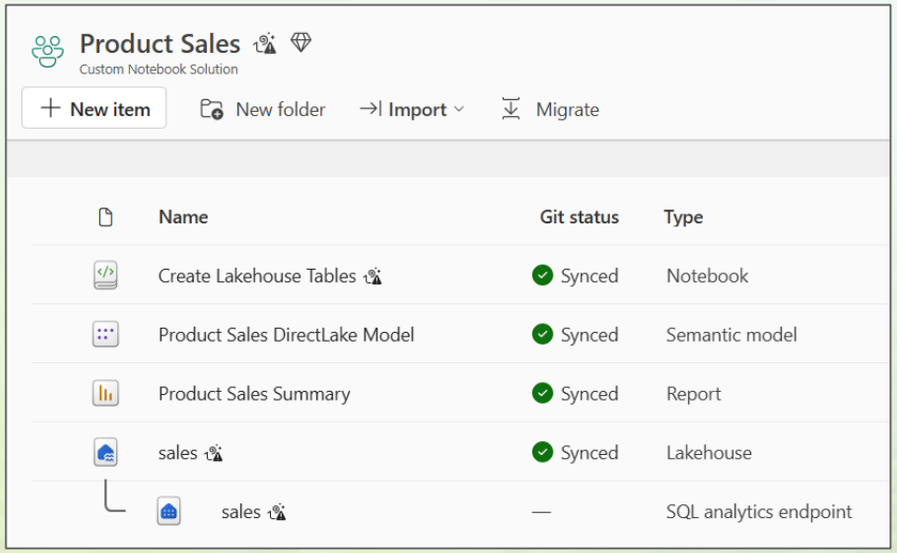
Enable Git Integration

Key benefits:

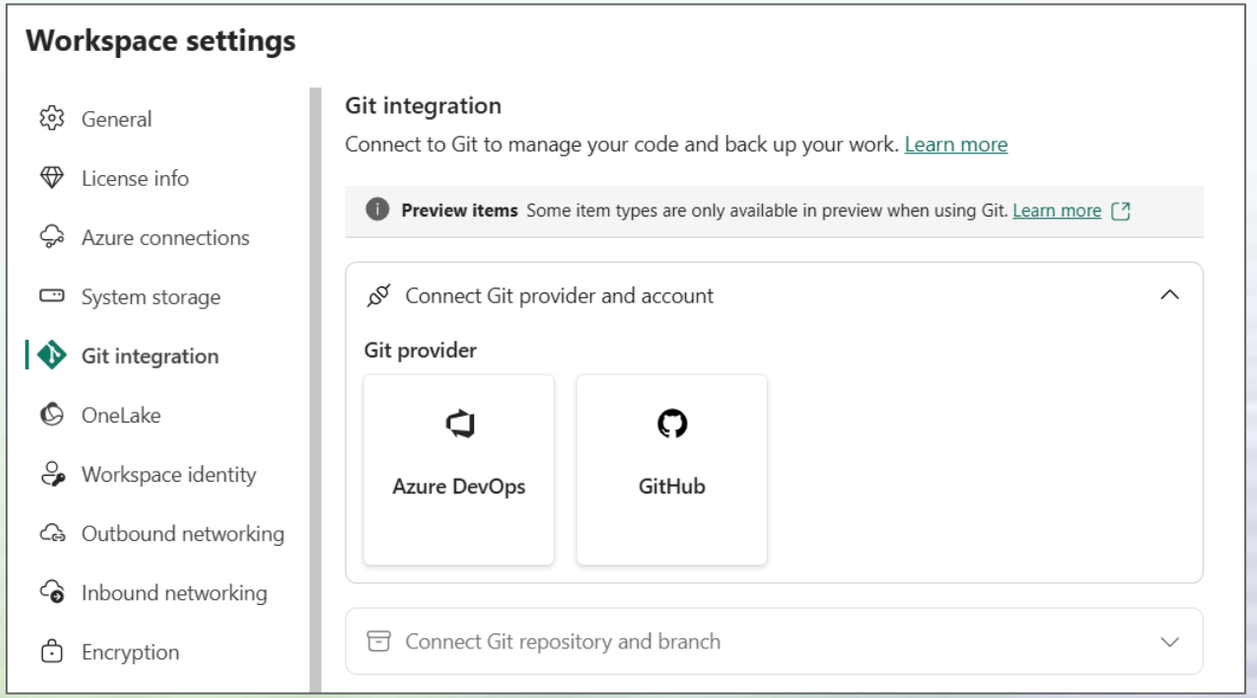
- track and have backups of the changes made by developers
- enable team collaboration according to the Git best practices

- Workspace items are building blocks used in any Fabric solution

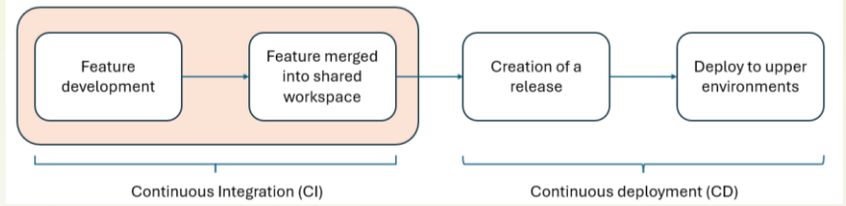




- Supported GIT providers: Azure DevOps, GitHub (others will be added in the future)
- GIT integration enabled at workspace scope

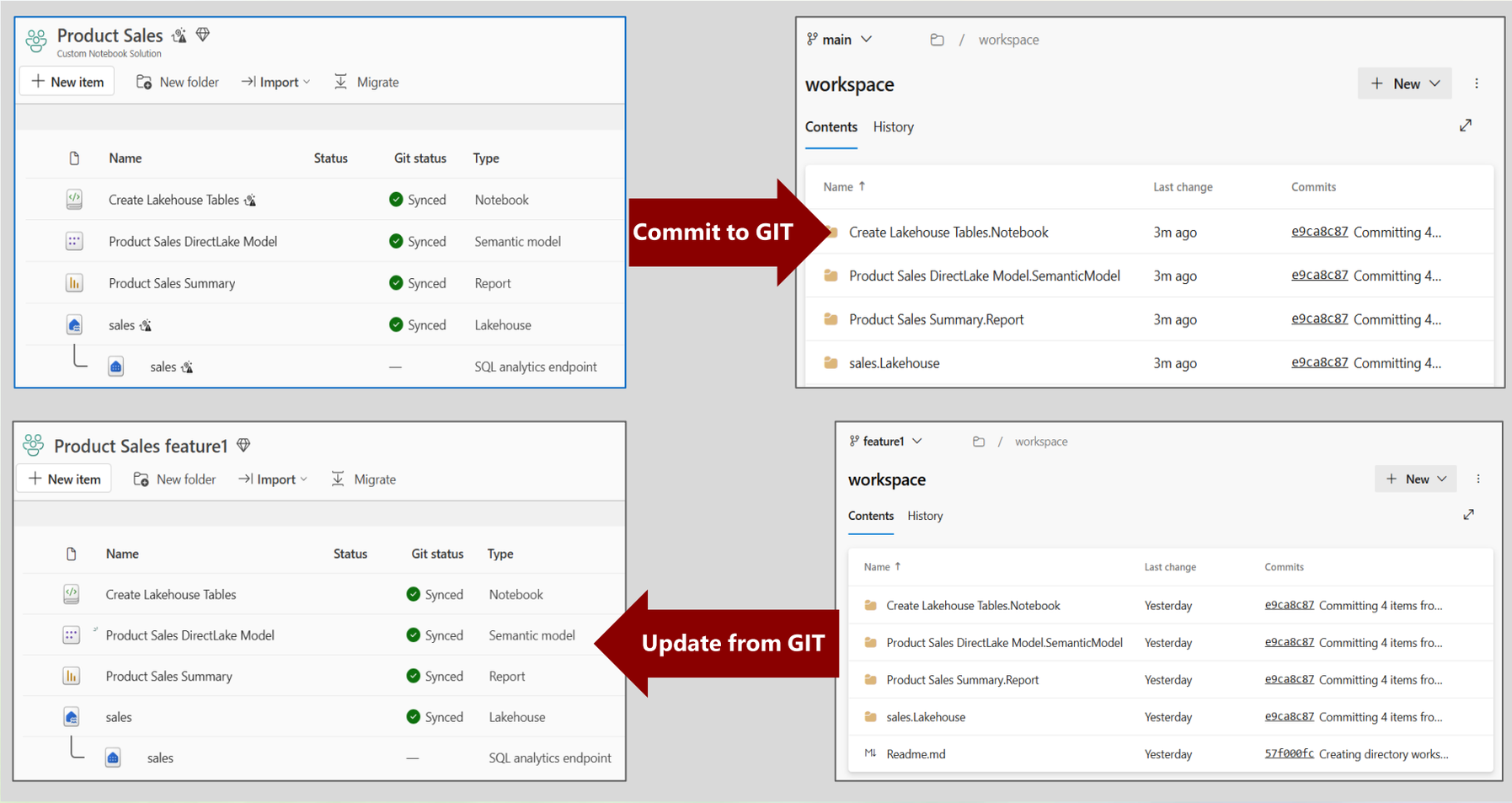


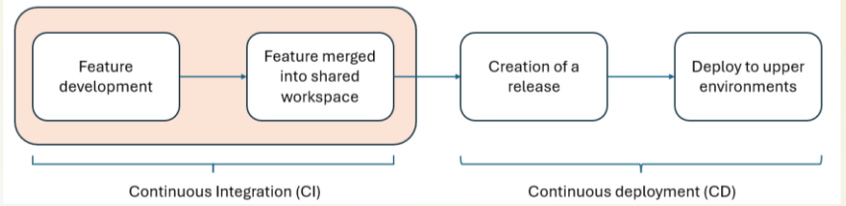
Enable Git Integration



- **Commit to Git** - When you connect your Fabric Workspace to a repository, Fabric's GIT integration serializes the workspace's item definitions and writes the definition files to the branch it is linked to

- **Update from Git -** On the other hand, Fabric's integration with GIT allows you to create/update workspace items from the item definitions already present in the GIT branch. Once synchronization is complete, the Git status of these items is displayed as "Synced"

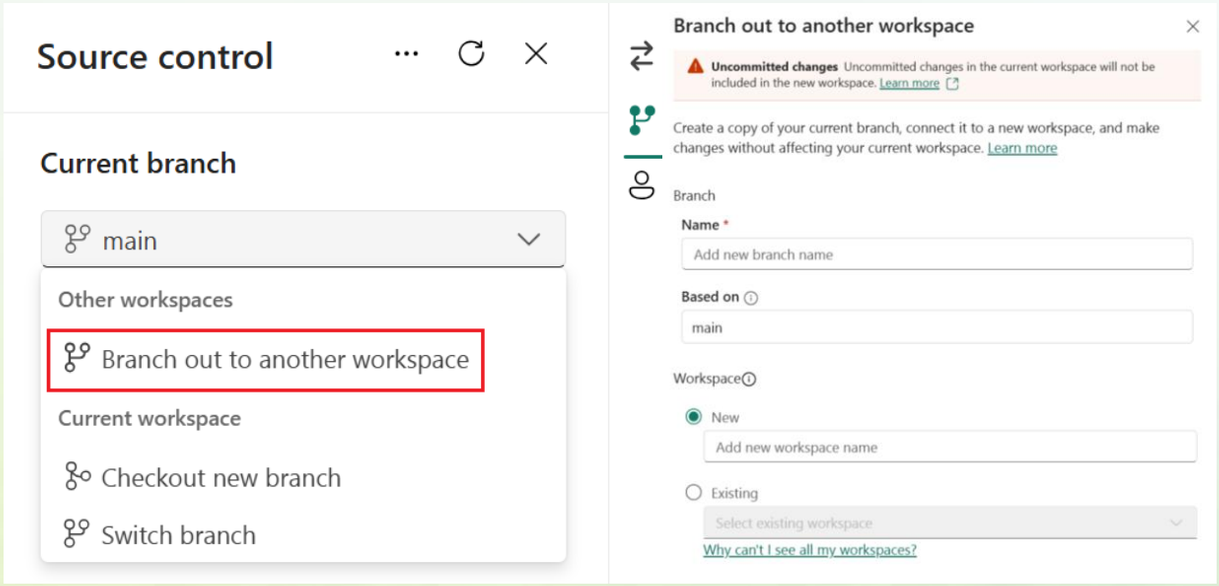




How to manage Feature Workspaces?

Feature workspaces provide recommended approach for development

Scenario #1A: create (short-lived) Feature Workspace from the UI

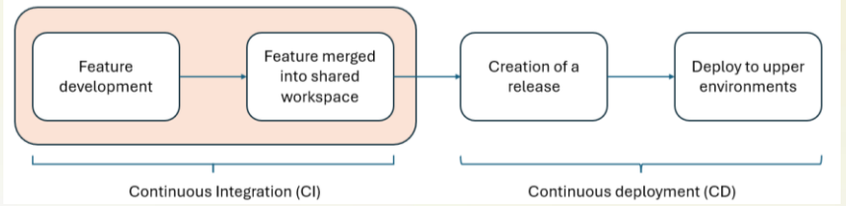


Scenario #1B: rely on the Terraform Provider for Microsoft Fabric to setup, configure and assign a (long-lived) Feature Workspace in advance


```
terraform {
  required_version = ">= 1.8, < 2.0"
  required_providers {
    fabric = {
      source  = "microsoft/fabric"
      version = "0.0.0-preview"
    }
  }
}

provider "fabric" {}

resource "fabric_workspace" "example" {
  display_name = "Example Workspace"
  description  = "Example Workspace Description"
}
```

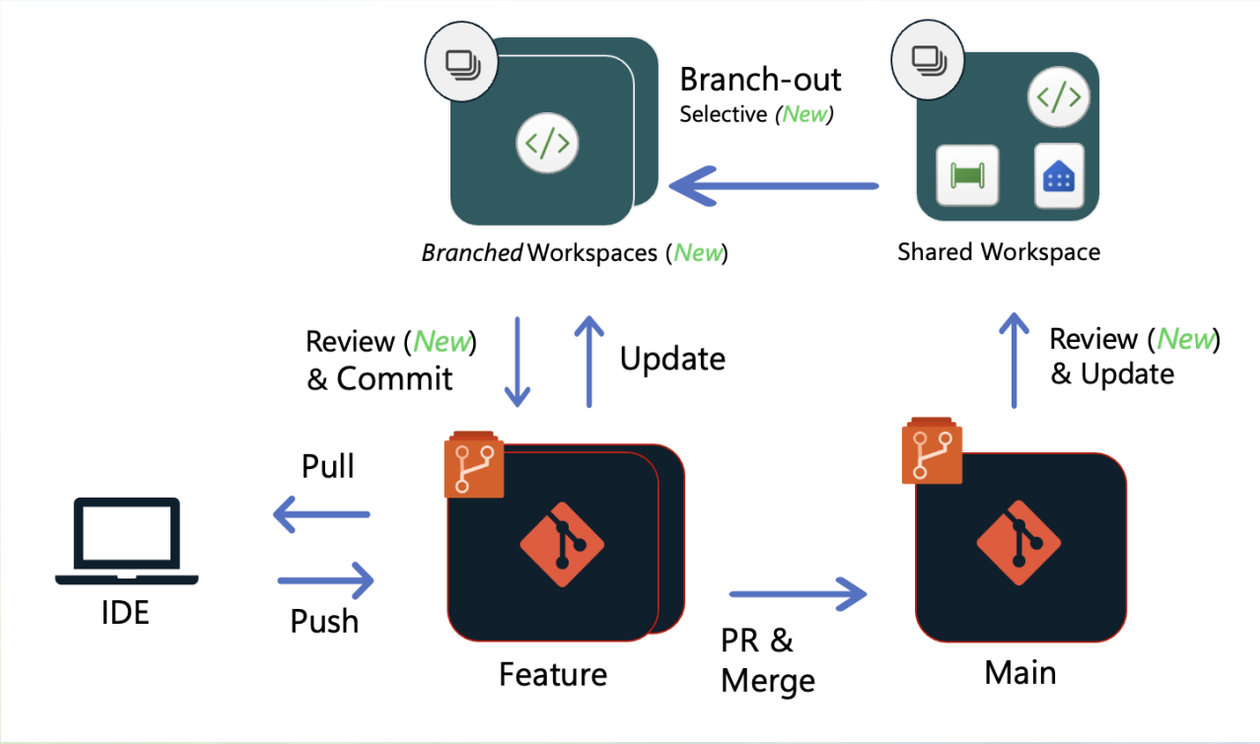


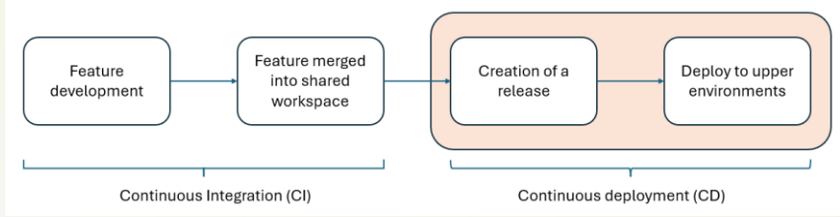
How to manage Feature Workspaces?

Feature workspaces provide recommended approach for development

Scenario #1C (according to the new Git developer experiences in Microsoft Fabric in Preview)

- rely on **Branched Workspaces**, which introduce a formal relationship between a source workspace and the workspaces created from it during a branch-out operation
- rely on **Selective Branching**, where developers can branch out with only the items they need, instead of copying an entire workspace

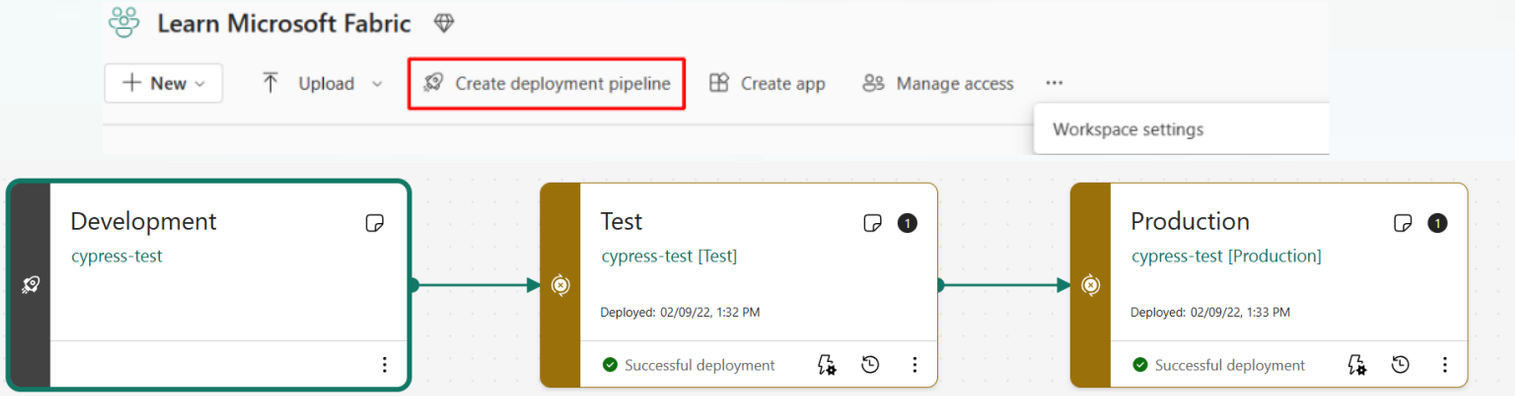


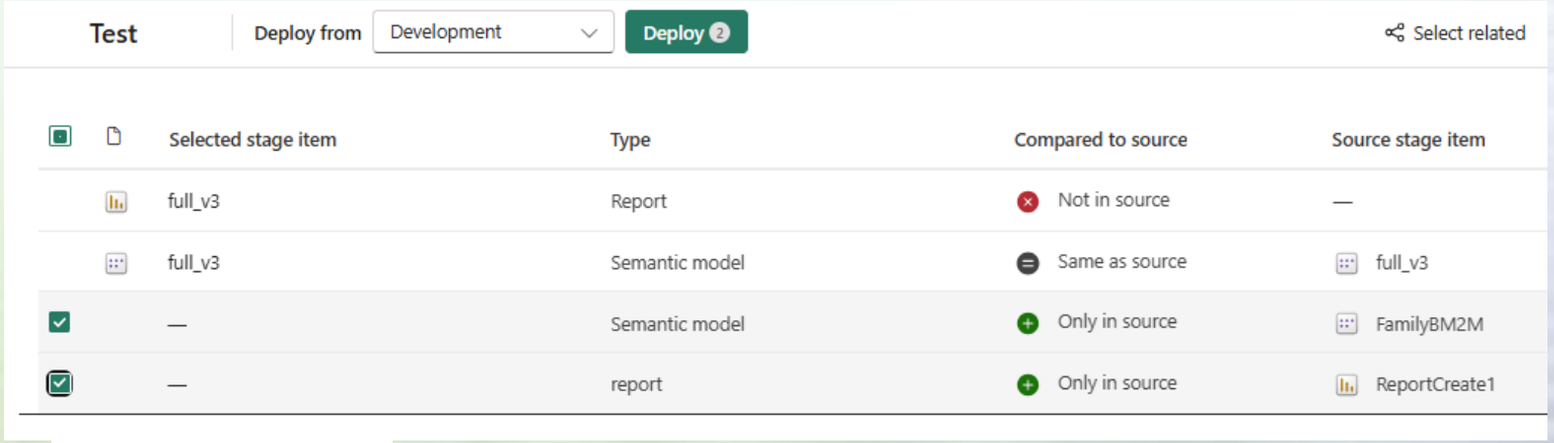


Fabric Deployment Pipeline

Capabilities:

- Configurable and manageable directly from the UI
- Manages the lifecycle of workspace items based on the concept of “stages” (each stage represents a specific environment linked to a Fabric workspace)
- Main options for deploying the content:
  - Full deployment (deploy all your content to the target stage)
  - Selective deployment (select which content to deploy to the target stage)





Scenario #1: Git collaboration for Data Engineers – what's next

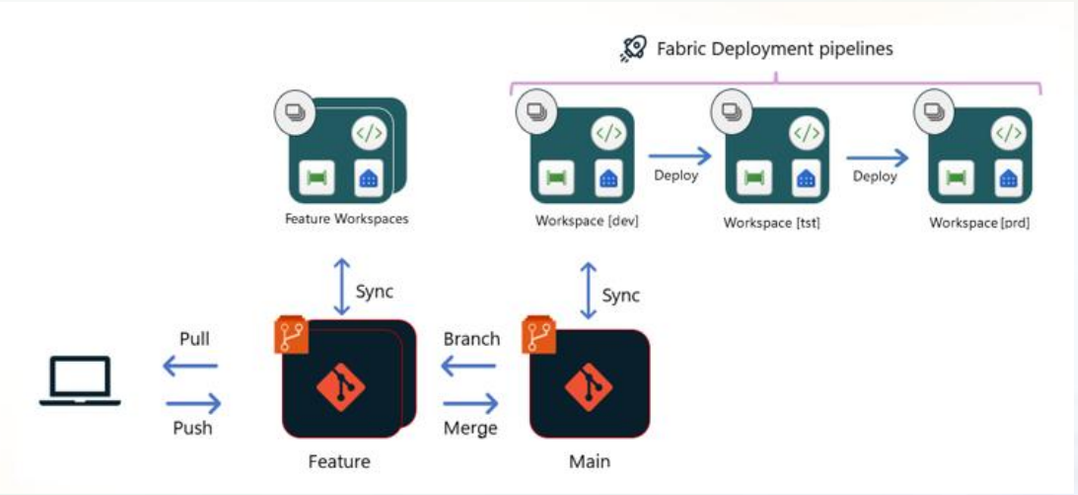
- **CI:** Git Integration & Feature workspaces
- **CD:** Fabric Deployment Pipelines (only from the UI)

Capabilities:

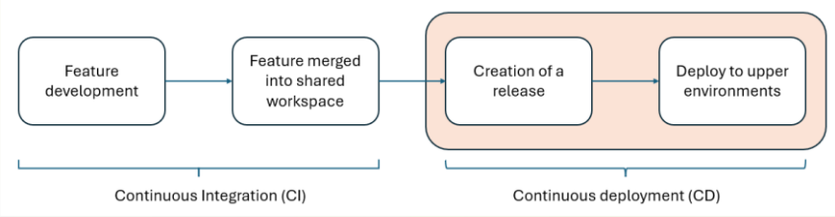
- backup and keep track of (almost) all the Fabric assets, allowing Git collaboration for Developers working on Data Engineering tasks

Drawbacks:

- lack of managing environment-specific configuration (limited parameterization options, e.g., connections, shortcuts, notebook parameters, data pipelines properties)
- lack of integration into an existing organization's automation process (e.g., in Azure DevOps Pipelines)
- limited Integration with 3rd party tools



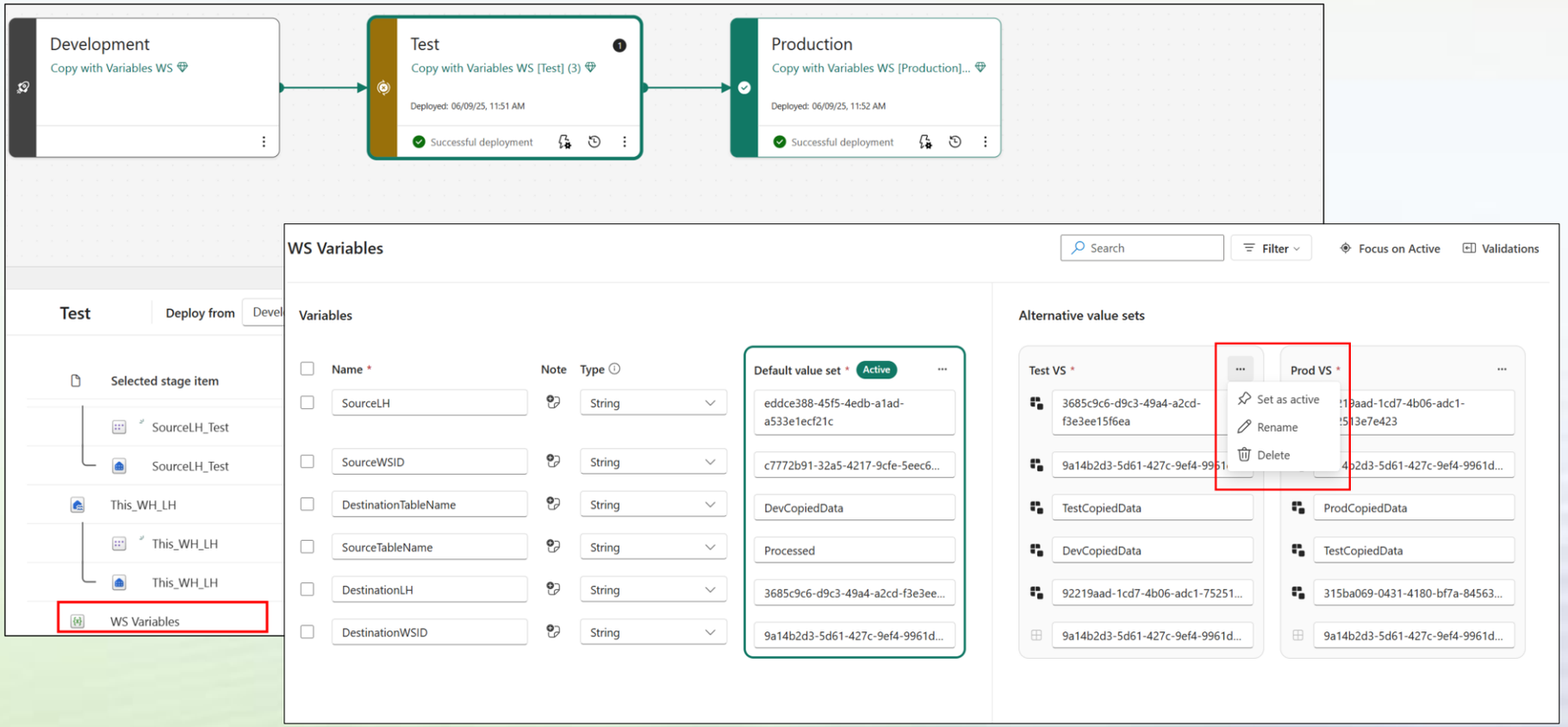
- rely on Variable Library or the **"Fabric-cicd" Python Library** (Scenario #3)
- **Scenario #2**: rely on Fabric Deployment Pipelines APIs
- **Scenario #6**: integration with 3rd party tools



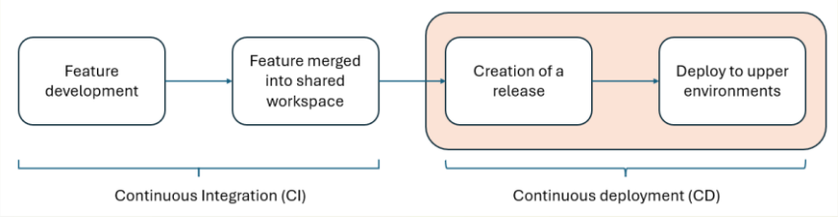
Variable Library

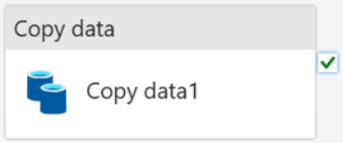
Capabilities:

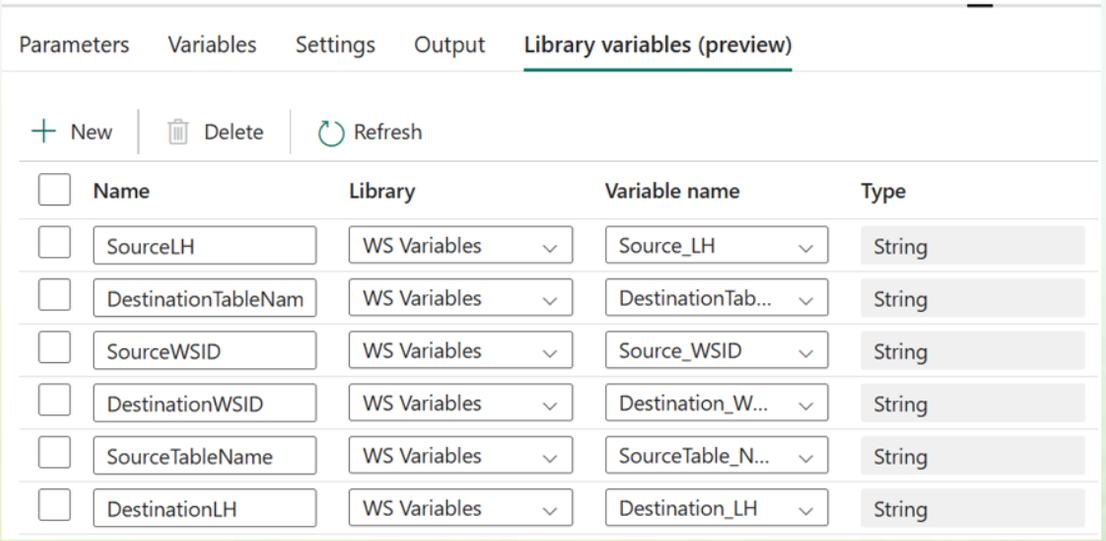
- Workspace item used to manage and share configuration parameters across various Fabric components
- Supported by different workspace items, such as Notebooks, Data Pipelines, Lakehouse shortcuts, and Dataflows Gen2
- Define exactly one active value set per workspace. Fabric items automatically consume the values from this active set during deployment

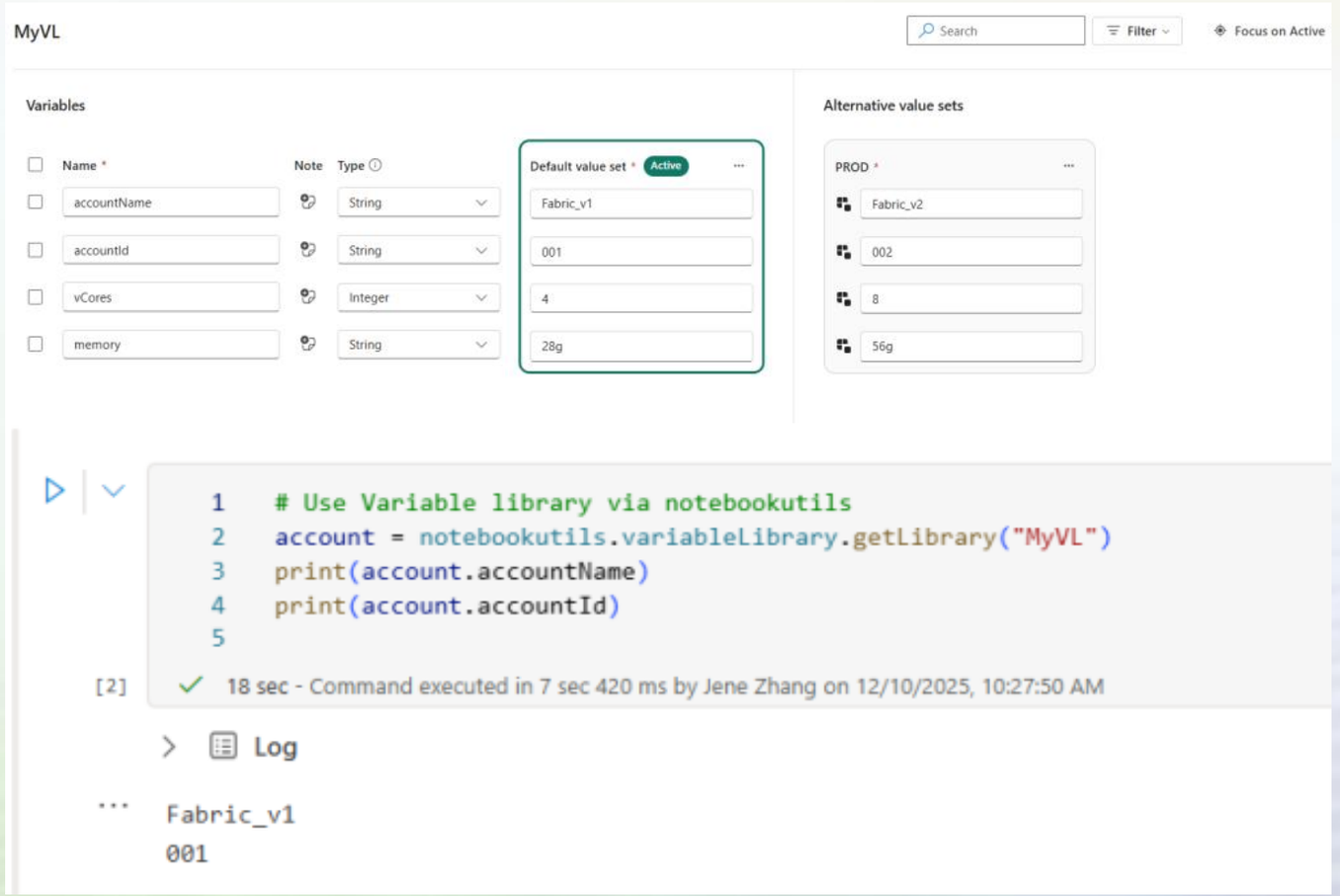


Variable Library







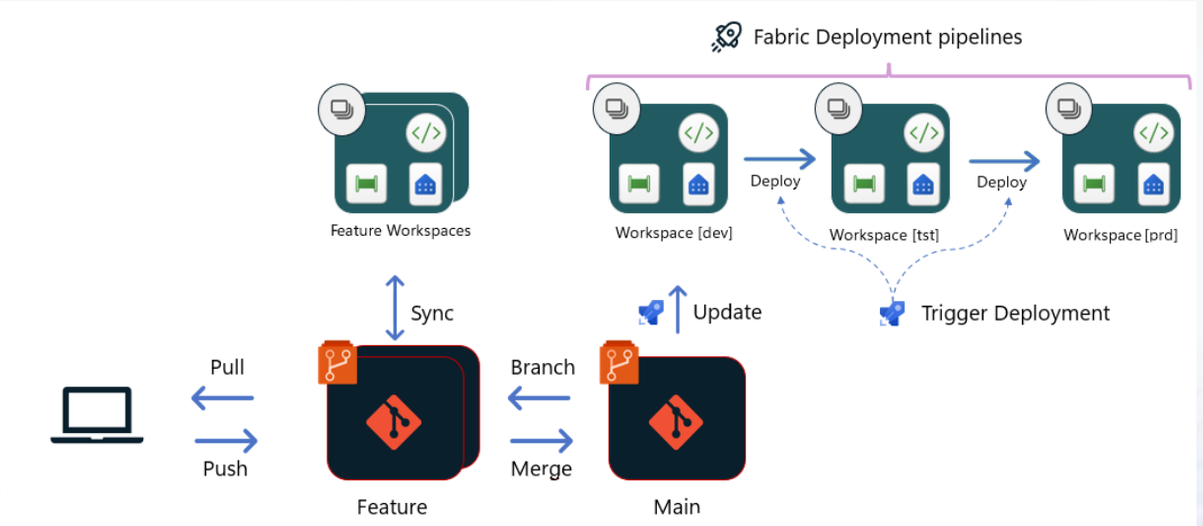


[Use Variable library in data pipelines]

[Use Variable library in notebook (GA since Dec 2025)]

Scenario #2: Orchestrate Fabric Deployment pipelines with ADO

- **CI:** Git Integration & Feature workspaces
- **CD:** Orchestrate the Fabric Deployment Pipelines from ADO Pipelines using Fabric REST APIs (docs)



Capabilities:

- backup and keep track of (almost) all the Fabric assets, allowing Git collaboration for the Developers
- manage and trigger the deployments from ADO Pipelines
- manage environment-specific configurations across stages using Variable Library

Drawbacks:

- Variable Library is limited to simple key-value pairs, it is not suitable for complex deployments via code which requires fully automated CI/CD process
- Limited Integration with 3rd party tools

- **Scenario #3**: rely on the **"Fabric-cicd" Python Library** for advanced CI/CD processes using YAML Pipelines
- **Scenario #6**: integration with 3rd party tools

improove

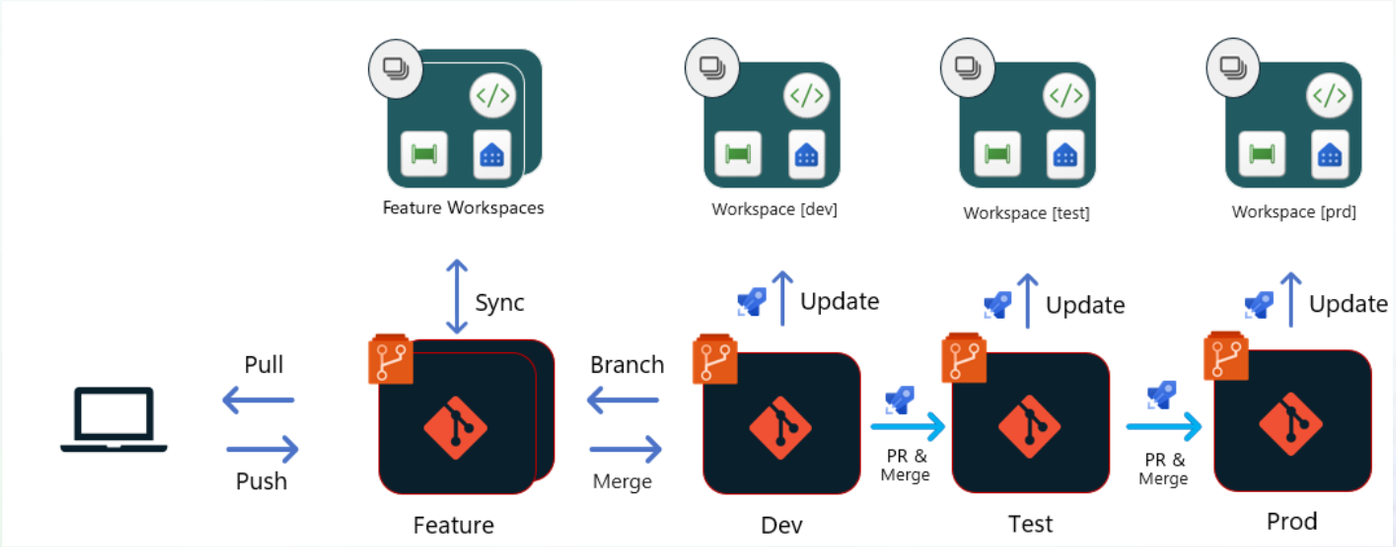
TD SYNNEX

Scenario #3: ADO Pipelines with the “fabric-cicd” Python Library

Use case: end-to-end **Medallion architecture** (Bronze, Silver, and Gold layers) to process raw data into actionable business insights

- **Data engineers** developing data integration and/or data processing workflows through Fabric Data Pipelines and Fabric Notebooks, that need flexible parameterization, dependency management and granular control
- **System Admin** requires the adoption of more robust CI/CD workflows, with advanced approval setting, into an existing organization’s automation suite (such as Azure DevOps)

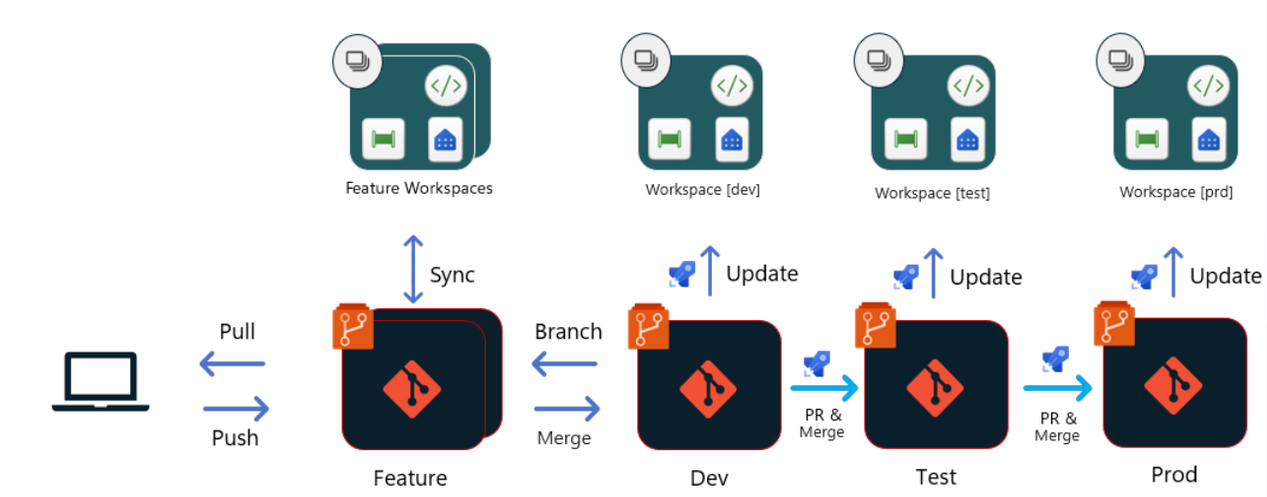
- **CI:** Git Integration & Feature workspaces
- **CD:** ADO YAML Pipelines

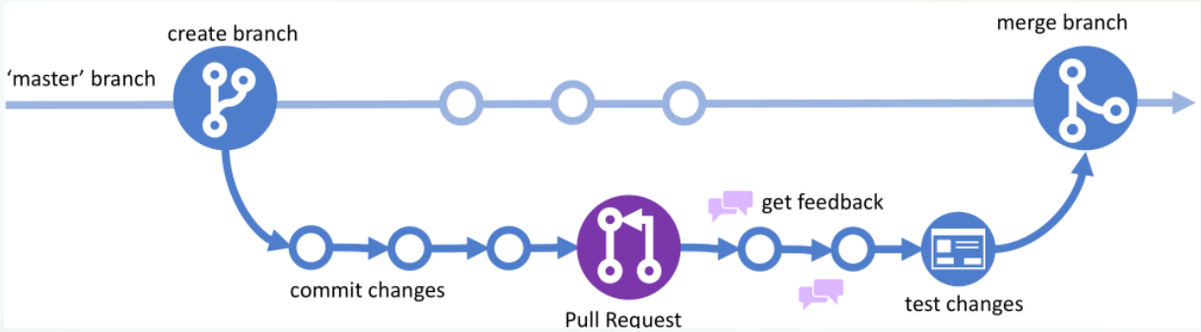


Capabilities:

- backup and keep track of (almost) all the Fabric assets, allowing Git collaboration between Developers
- robust management of the deployments using ADO Pipelines
- manage environment-specific configurations (“parametrization”) using the “parameter.yml” file of the Python Library

Scenario #3: the overall workflow





1. Setup in advanced a **Feature workspace** for each developer (probably many, so do it programmatically!)
2. Enable GIT connection in the **Feature workspace** and then create a **«feature» branch** managed by the developer who owns the workspace. This is a short-lived branch that must be deleted after a Pull Request in the «dev» branch.
   Repeat for each feature workspace
3. **Submit a PR to merge changes into the «dev» branch** (protect this branch with branch policy to prevent direct commits and pushes)
4. On PR approval, the **ADO YAML Pipeline is triggered**. After optionally passing any manual approval gate, it deploys these updates to the **Fabric DEV workspace** (the "dev" branch acts as source of truth for the DEV workspace, but is not attached to the Workspace)
5. Repeat the process creating new PRs to promote the Fabric assets up to the Fabric PRD workspace

improove

TD SYNNEX

“fabric-cicd” Python Library

since February 2026 is officially supported by Microsoft

Capabilities:

- Code-first solution for deploying Fabric items from a Git repository into a target workspace
- Manage environment-specific configurations (“parametrization”) using the “parameter.yml” file
- Service Principal AuthN method supported

Installation

Run the following shell command

```
pip install fabric-cicd
```

Configuration

Create the following Python script

```python
from fabric_cicd import FabricWorkspace, publish_all_items, unpublish_all_orphan_items

# Initialize the FabricWorkspace object with the required parameters
target_workspace = FabricWorkspace(
    workspace_id = "your-workspace-id",
    repository_directory = "your-repository-directory",
    item_type_in_scope = ["Notebook","DataPipeline","Environment"],
)

# Publish all items defined in item_type_in_scope
publish_all_items(target_workspace)

# Unpublish all items defined in item_type_in_scope not found in repository
unpublish_all_orphan_items(target_workspace)
```

An exhaustive example of all capabilities currently supported in the `parameter.yml` file.

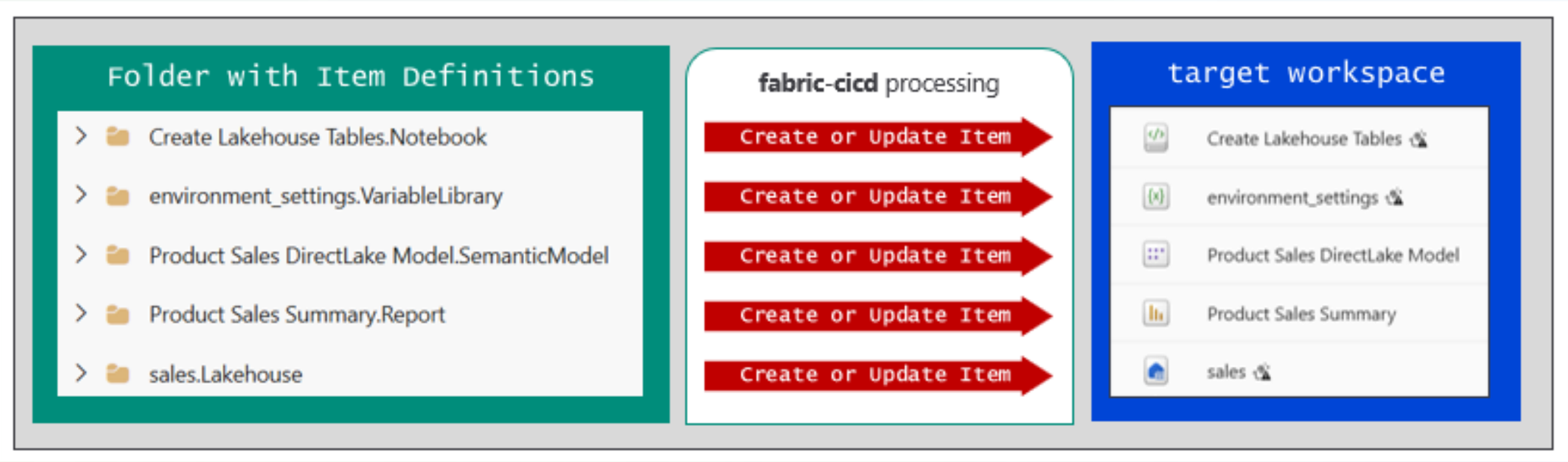
```yaml
find_replace:
    - find_value: "123e4567-e89b-12d3-a456-426614174000" # lakehouse GUID to be replaced
      replace_value:
          PPE: "f47ac10b-58cc-4372-a567-0e02b2c3d479" # PPE lakehouse GUID
          PROD: "9b2e5f4c-8d3a-4f1b-9c3e-2d5b6e4a7f8c" # PROD lakehouse GUID
      item_type: "Notebook" # filter on notebook files
      item_name: ["Hello World", "Goodbye World"] # filter on specific notebook files
      file_path:
    - find_value: "8f5c0cec-a8ea-48cd-9da4-871dc2642f4c" # workspace ID to be replaced
      replace_value:
          PPE: "d4e5f6a7-b8c9-4d1e-9f2a-3b4c5d6e7f8a" # PPE workspace ID
          PROD: "a1b2c3d4-e5f6-7a8b-9c0d-1e2f3a4b5c6d" # PROD workspace ID
      file_path: # filter on notebook files with these paths
          - "/Hello World.Notebook/notebook-content.py"
          - "\\Goodbye World.Notebook\\notebook-content.py"
```

“fabric-cicd” Python Library

since February 2026 is officially supported by Microsoft

Capabilities:

- Code-first solution for deploying Fabric items from a Git repository into a target workspace
- Manage environment-specific configurations (“parametrization”) using the “parameter.yml” file
- Service Principal AuthN method supported

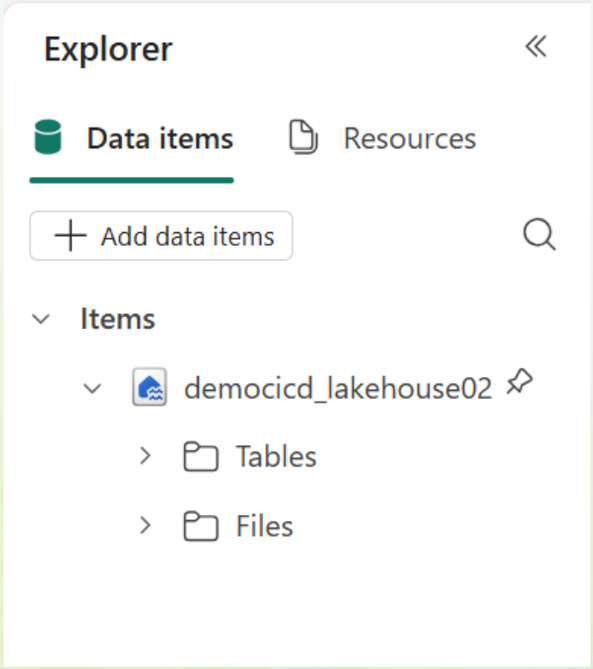


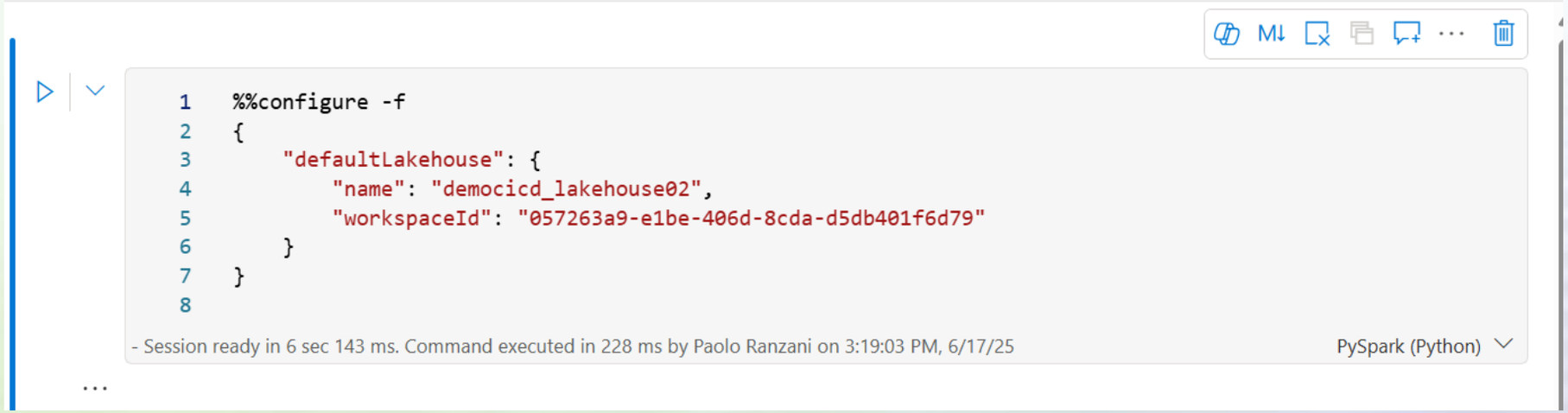
“fabric-cicd” Python Library

since February 2026 is officially supported by Microsoft

Good practices:

- NO (left): manually connect a notebook to a Lakehouse
- YES (right): programmatically connect a notebook to a Lakehouse. These metadata will then be overwritten in the deployment stage with the information specified in the «parameter.yml» file (e.g., the workspace ID of DEV will be replaced by the target workspace ID of QLT or PRD, that must be created in advanced)



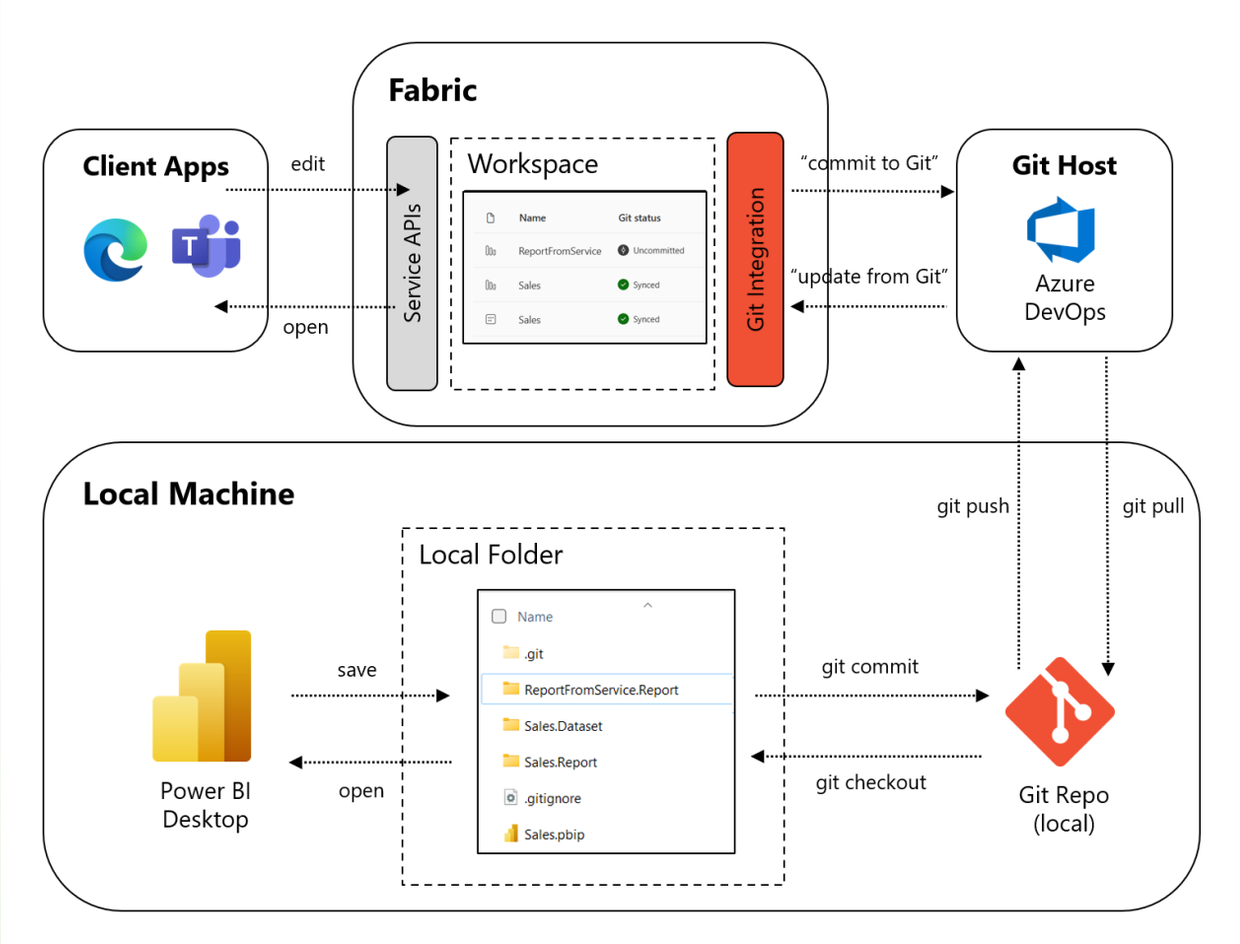


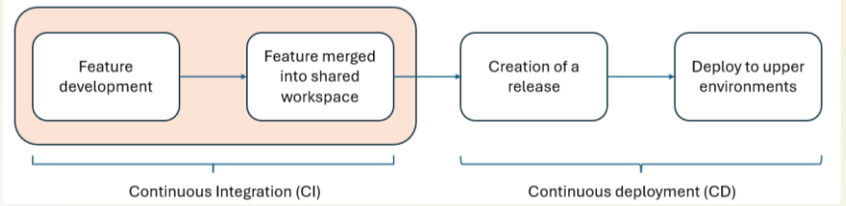
Scenario #4: Git Collaboration for PBI Developers

Use case: PBI Developers are tasked with adding new features to different shared executive reports, where other colleagues are also making changes

- **PBI developers** implementing PBI reports, dashboards and semantic models

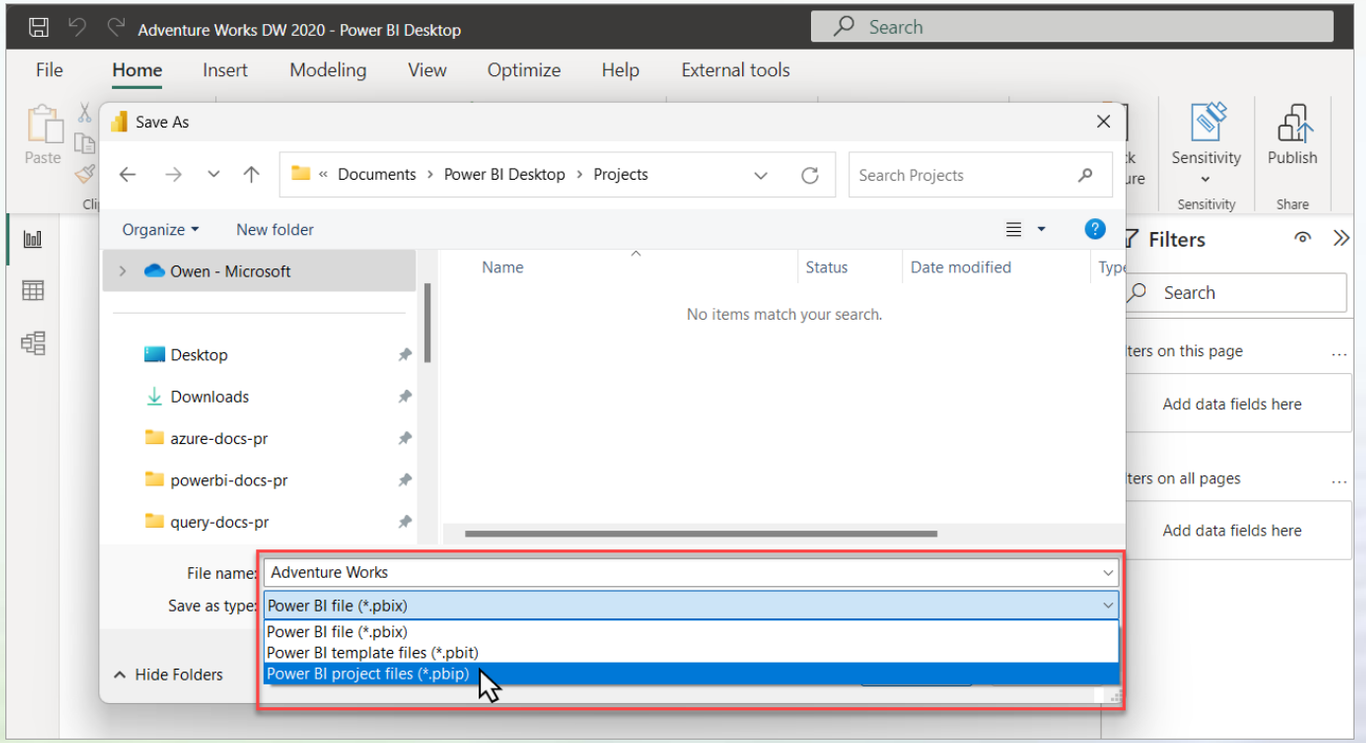
- **CI:** IDE (Power BI Desktop using PBIP format) & Git Integration
- **CD:** Fabric Deployment Pipelines

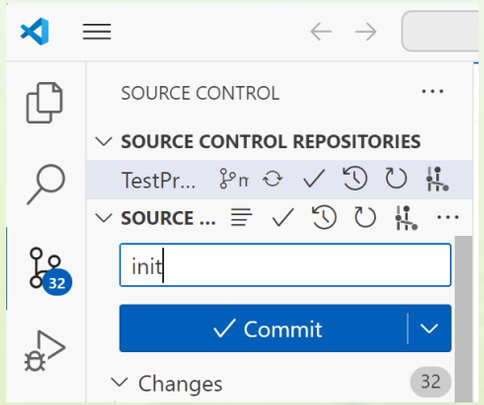


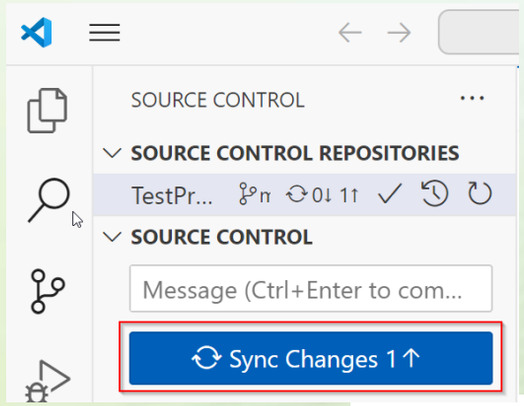


Scenario #4: CI using PBIP format

1. In Power BI Desktop, save your work as a **Power BI project file (pbip)** format
2. In VS Code Git Integration, **any changes you make in Power BI Desktop changes a file in the folder tracked by your local Git**. Commit and push (sync) changes up to the remote Git Repo
3. In **the Fabric Workspace, pull the changes from the ADO Repos** to update the contents



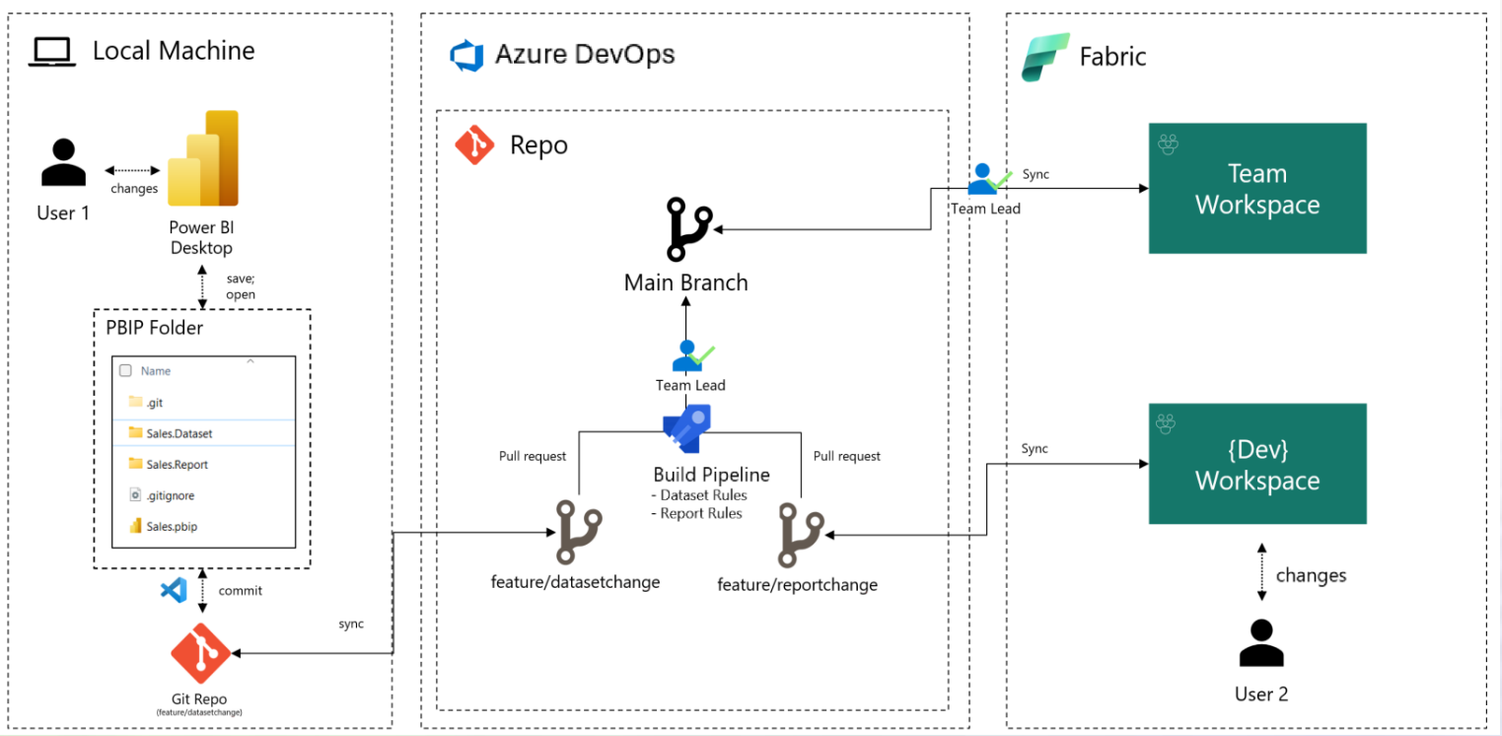




Scenario #5: scenario #3 + scenario #4 with Power BI quality checks

Use case: end-to-end **Medallion architecture** (Bronze, Silver, and Gold layers) to process raw data into actionable business insights

- **Data engineers** developing data integration and/or data processing workflows through Fabric Data Pipelines and Fabric Notebooks, that need flexible parameterization, dependency management and granular control
- **System Admin** requires the adoption of more robust CI/CD workflows, with advanced approval setting, into an existing organization's automation suite (such as Azure DevOps)
- **PBI Developers** implementing PBI reports, dashboards and semantic models

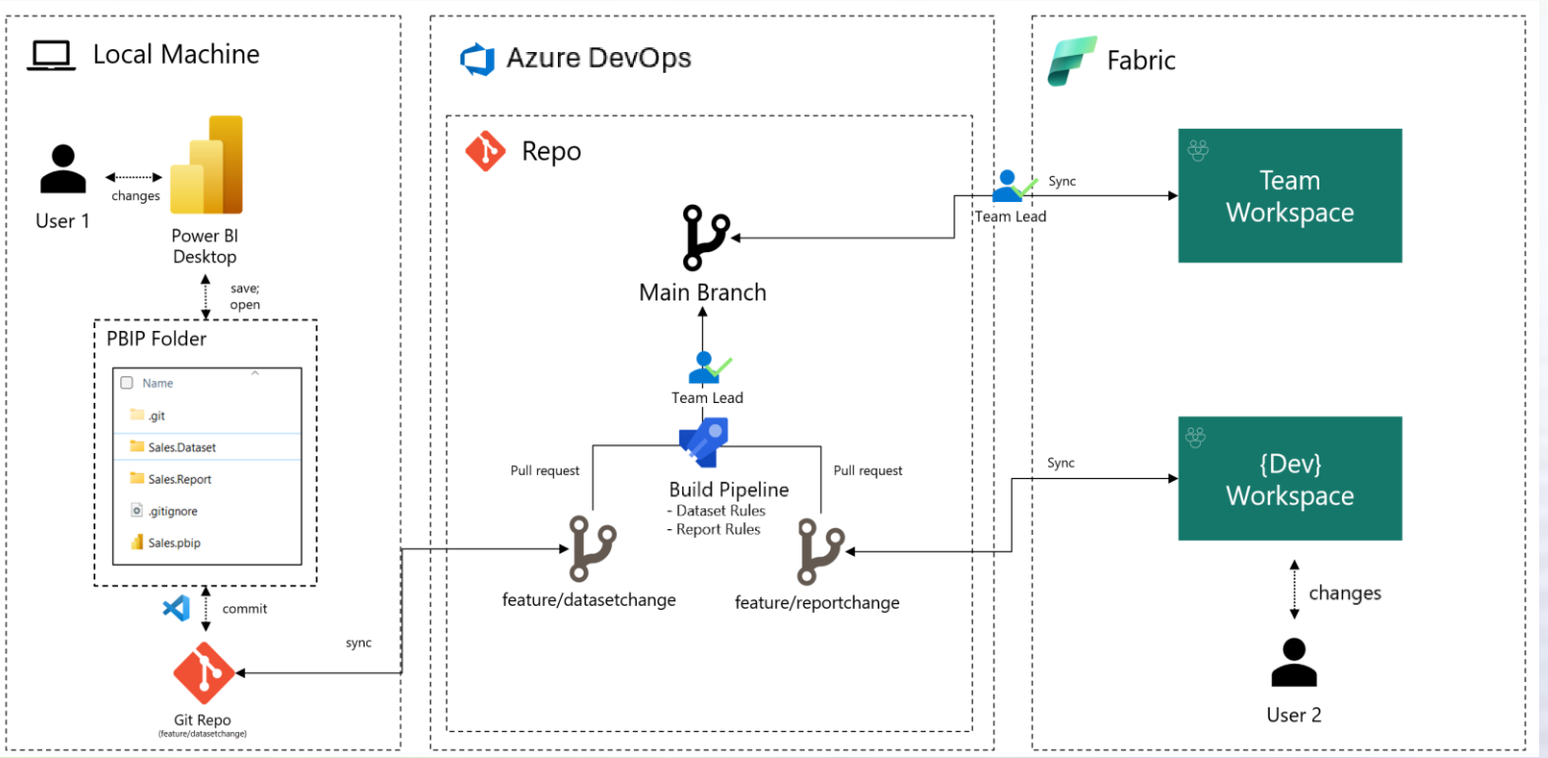


- **CI:** Feature workspaces for both Data Engineers and PBI Developers + **ADO Build Validation Pipelines using BPA/PBI Inspector**
- **CD:** ADO YAML Pipelines using "fabric-cicd" Python Library

Scenario #5: scenario #3 + scenario #4 with Power BI quality checks

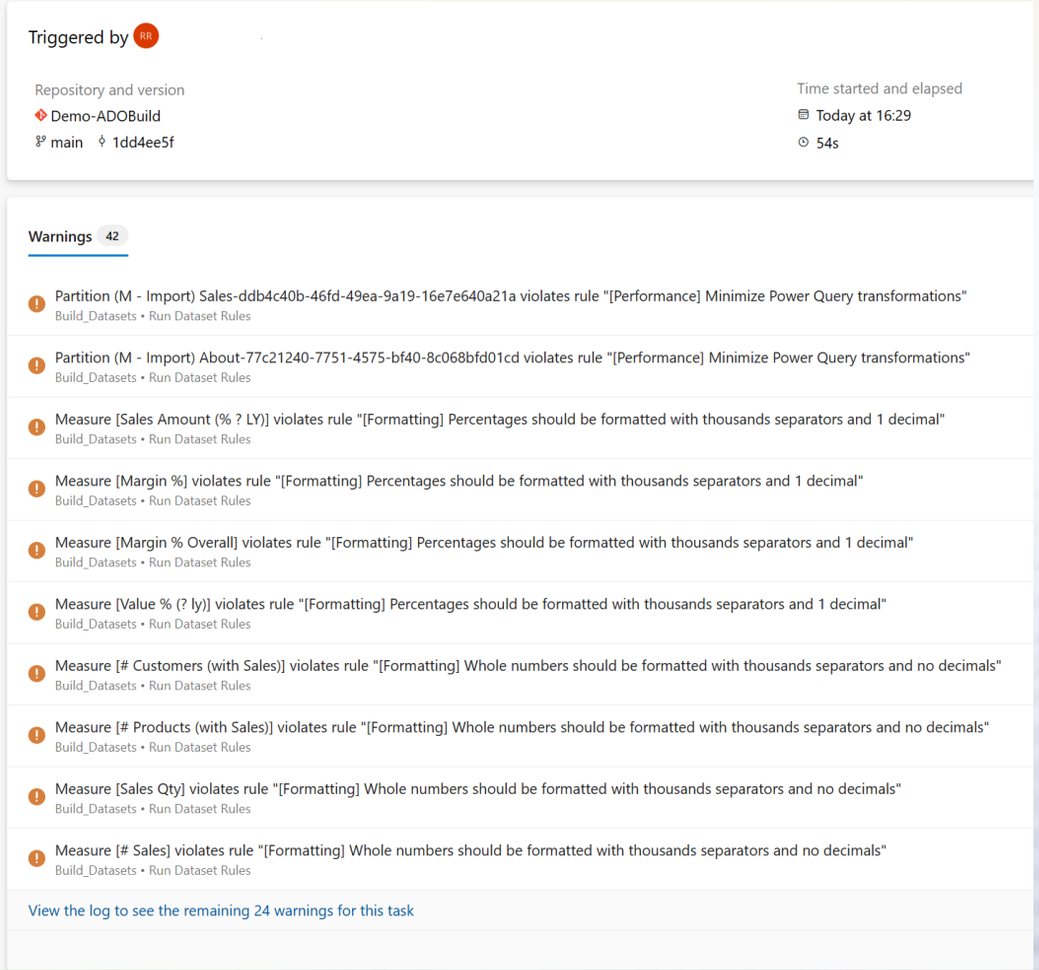
Capabilities:

- backup and keep track of (almost) all the Fabric assets, allowing Git collaboration for the Developers
- robust management of the deployments using ADO Pipelines
- manage environment-specific configurations (“parametrization”) using the “parameter.yml” file of the “fabric-cicd” Python Library
- Build Validation pipelines to automatically perform rule-based checks on datasets/reports using BPA/PBI Inspector



PBI quality checks in ADO using BPA/PBI Inspector

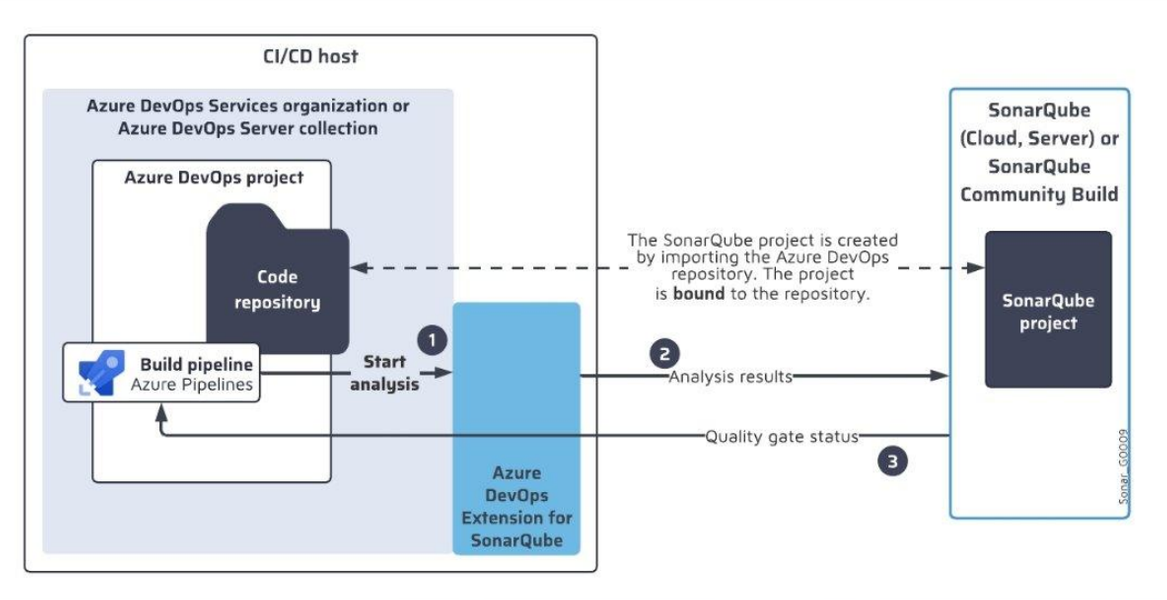
- **What**: automatically perform rule-based checks on the metadata of Semantic Models and PBI Reports when a new Pull Request is created
- **How to**: use an ad-hoc Azure DevOps Pipeline ("validation pipeline") that, via Branch Policy, is triggered by every Pull Request to a given target branch (enforced by approval gates if required).
  - If OK -> merge from source branch (any feature branch) to the target branch (e.g. the "Dev" branch)
  - If KO -> PBI developer must fix the errors on their Feature Workspace and submit a new PR
- **Tools:**
  - **Best Practice Analyzer (BPA)** - set of predefined checks that analyze the metadata of a Power BI semantic model to identify potential issues
  - **PBI Inspector** - Ensure that your reports follow best practices for consistency, performance, and accessibility by applying customizable rules to report metadata



Scenario #6: scenario #5 + 3rd party integrations (SonarQube)

Use case: end-to-end **Medallion architecture** (Bronze, Silver, and Gold layers) to process raw data into actionable business insights

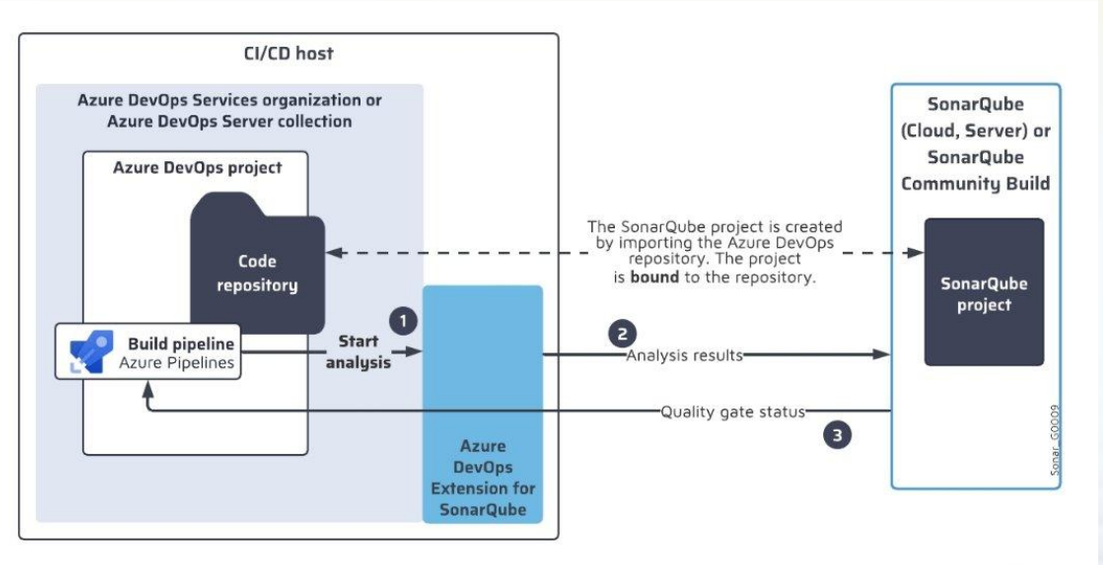
- **Data engineers** developing data integration and/or data processing workflows through Fabric Data Pipelines and Fabric Notebooks, that need flexible parameterization, dependency management and granular control
- **System Admin** requires the adoption of more robust CI/CD workflows, with advanced approval setting, into an existing organization's automation suite (such as Azure DevOps), **also performing Static Code Analysis (e.g. using SonarQube)**
- **PBI Developers** implementing PBI reports, dashboards and semantic models



- **CI:** Feature workspaces for both Data Engineers and PBI Developers + ADO Build Validation Pipelines using BPA/PBI Inspector + **ADO ADO Build Validation Pipelines to perform Static Code Analysis** (detect coding issues before deploy any .py or .sql artifact)
- **CD:** ADO YAML Pipelines using "fabric-cicd" Python Library

Scenario #6: additional extensions

- **Performing Static Code Analysis using SonarQube**
  - Detect coding issues before deploy a software artifact (e.g., any .py or .sql files)
  - Requires the Azure DevOps Sonarqube extension



- **Improving your change management process using ServiceNow**
  - allow an Azure DevOps pipeline to wait for an approved change request in ServiceNow before continuing the deployment
  - Requires the Azure DevOps ServiceNow extension

Stay tuned!

Docs

Microsoft **Fabric Roadmap** Forums Inspiration Ideas Communities Blogs Learning Support

- Administration, Governance and Security
- Cosmos DB (NoSQL)
- Data Engineering
- Data Factory
- Data Science
- Data Warehouse
- **Fabric Developer Experiences**
- Fabric Ecosystem
- IQ
- OneLake

Planned — All features planned
Try Now — All features recently released
All — All features planned and released

Multiple task flows in a workspace	Planned	General availability Q1 2026
ADO Pipelines Extension for MS Fabric	Planned	General availability Q1 2026
Job alerts - notifications for failed jobs	Planned	General availability Q1 2026

ADO Pipelines Extension for MS Fabric

The Microsoft Fabric ADO Pipelines Extension bundles the Fabric CLI and enables automated workspace creation and deployment of Fabric items in Azure Pipelines, ensuring consistent and repeatable delivery across environments.

Release Date: Q1 2026

Release Type: General availability

improove

TD SYNNEX

Wrap up: the main characters

- **VCS**
  - Git
  - Azure DevOps Repos
  - Azure DevOps Pipelines
  - (GitHub repository and GitHub Actions)
- **Open source and 3rd-party tools**
  - “fabric-cicd” Python library (now officially supported by Microsoft)
  - Best Practices Analyzer
  - PBI Inspector
  - Terraform – Fabric plugin
  - Sonarqube and its ADO extension
  - ServiceNow and its ADO extension
- **Microsoft Fabric**
  - Workspaces (“Shared workspaces” vs “Feature workspaces” vs “Branched workspaces”)
  - Child items (notebooks, data pipelines, reports, lakehouses, …)
  - Fabric Deployment Pipelines
  - Variable Library

improove

TD SYNNEX

Take-home message

Feedbacks

Microsoft Fabric lifecycle management in enterprise scenarios

«Sapersi usare»
(both among tools and among teams!)

#GLOBALAZURE

improove

TD SYNNEX

#Milano

Slide e video:
https://www.globalazuremilano.it