DevOps Automation (CI/CD) for Microsoft Fabric

Agenda

📘 Introduction & Overview

- What is CI/CD?
- Features in Fabric
- Guidance and Roadmap

🏛️ Architecture Considerations

- Workspaces and Environments
- Medallion Architecture in Fabric

🔧 Continuous Integration (CI)

- Git Integration Demo

🚀 Continuous Deployment (CD)

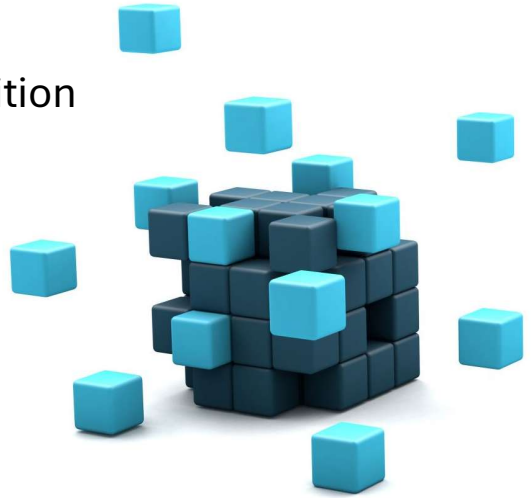
- Fabric Deployment Pipelines Demo
- Deploy Data and Connections
- Managing Data Connections Demo

What is CI/CD?

Continuous Integration: Merging code within an environment
Continuous Deployment: Releasing and promoting code between environments

Why?

- Reduce risk (lost code, failed releases) via increased reliability and repetition
- Allows team members to share, re-use, and jointly develop
- Improves security and traceability of both data and releases



Key Concept

"As-code" mentality ("Cattle, not pets") that idealizes the ability to create and destroy environments at will, reducing risks of code or data loss and dependency on a single developer.

CI/CD Features in Fabric

In the box (still evolving)

- Git Integration
- Deployment Pipelines
- Fabric REST API
- Microsoft Guidance: CI/CD workflow options in Fabric - Microsoft Fabric | Microsoft Learn
- Microsoft Tutorials: Lifecycle management tutorial - Microsoft Fabric | Microsoft Learn

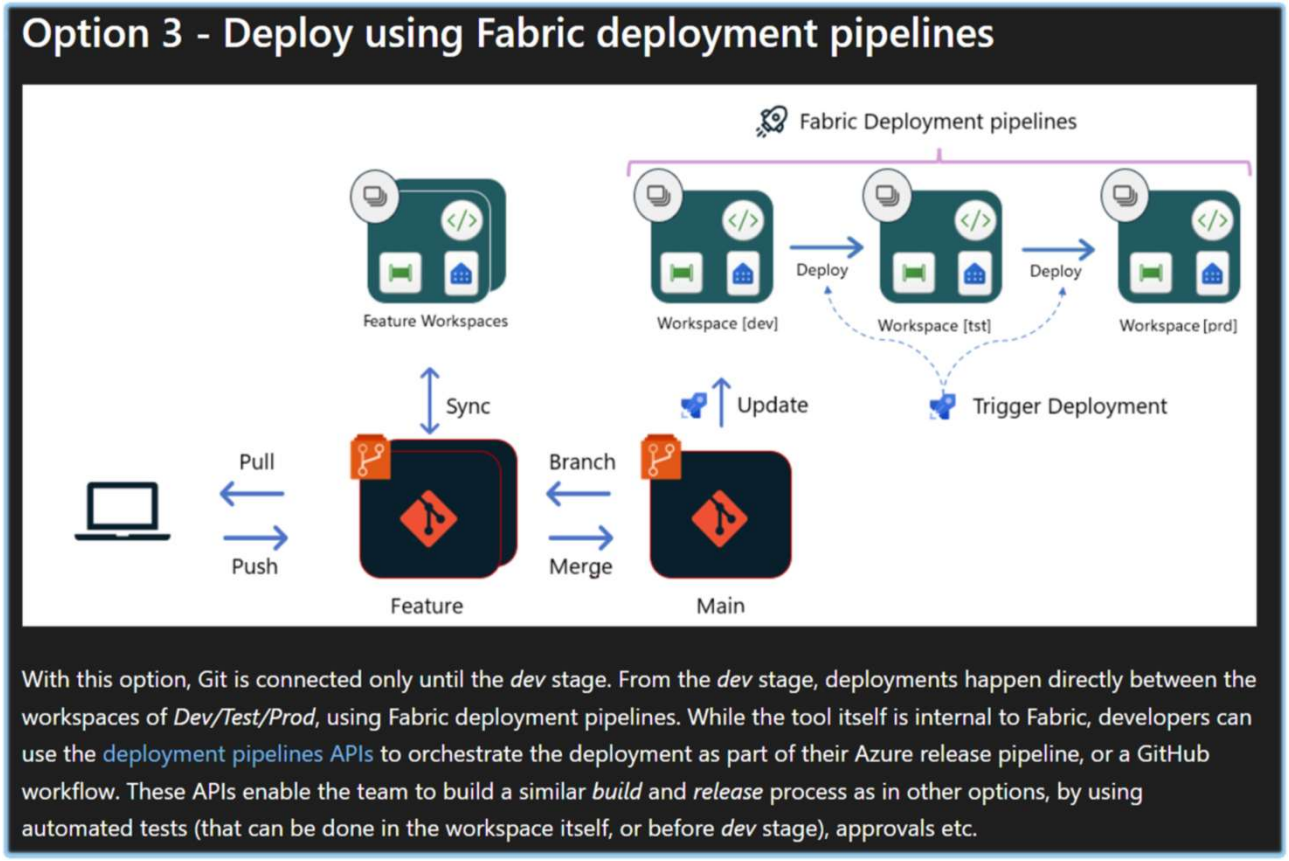
What else you'll need

- Azure DevOps or GitHub account
- Optional: Open-source python libraries and YAML templates, other CI/CD tools

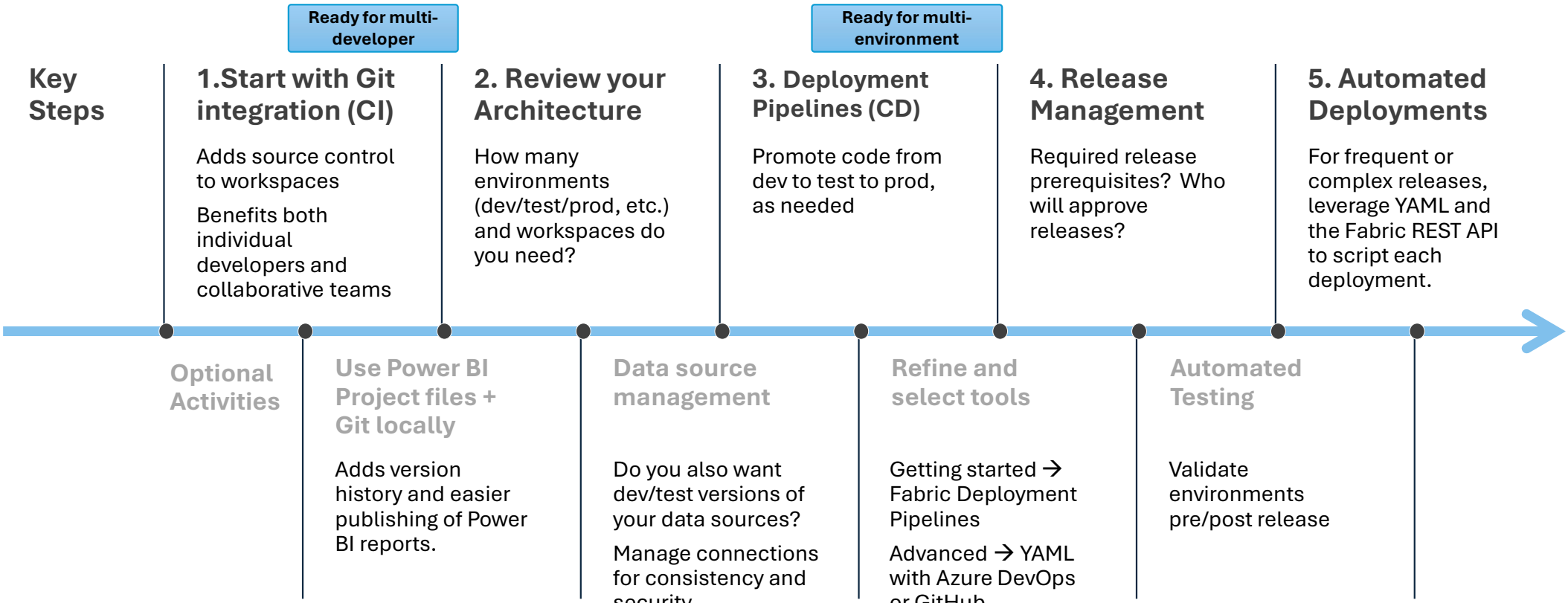
Microsoft Guidance: Four Deployment Options

Reference:
https://learn.microsoft.com/en-us/fabric/cicd/manage-deployment

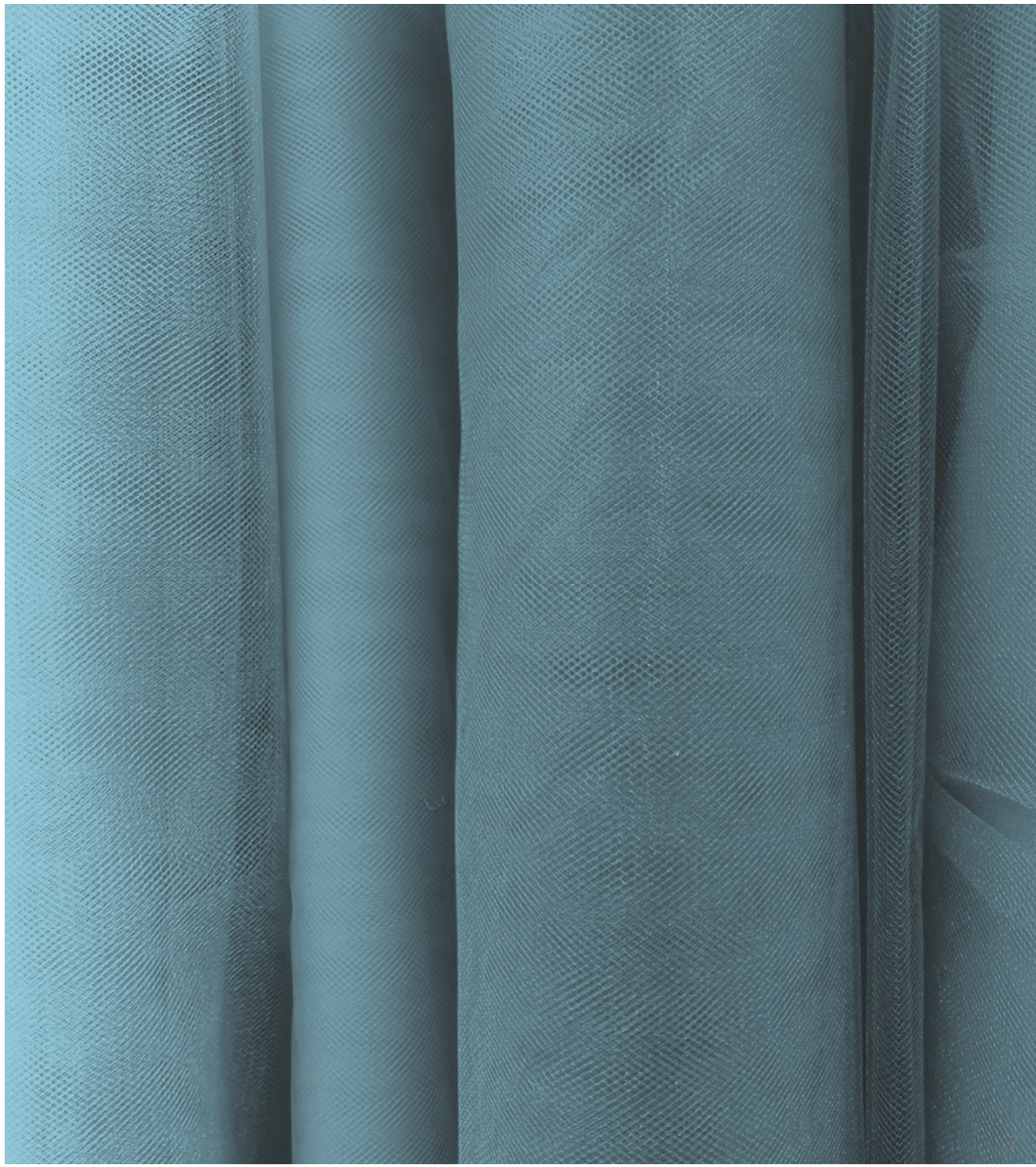
- The options are not mutually exclusive, mix-and-match as needed
- This demo will focus on Option 3, as the simplest starting point. This option meets most needs using "in-the-box" tools.



Suggested Adoption Path



Demo Prerequisites



Fabric Capacity

Azure DevOps (ADO) Account

- “Basic” license for each developer
- Free version of ADO is sufficient for small teams

For local Power BI Desktop Development

- Microsoft Power BI Desktop
- Microsoft Visual Studio Code
- Open-source Git install

Step 1: Start with Git Integration (CI)

In CI/CD, a Repository (repo) is a managed storage location for code, configuration, and Fabric items.

Repos are the foundation for all CI/CD workflows

In a Fabric, a Git repo maps 1:1 to a Workspace

Fabric offers native support for Git repos stored in either Azure DevOps or GitHub

Basic features (single developer)

- Synchronize workspaces with Git repositories
- Add source control and version history to Fabric/Power BI Development

Features for collaboration:

- Facilitate multiple developers working within a single workspace or the same Fabric item
- Create branches for feature development
- Add Pull Requests (PRs) and peer reviews

Demo Overview: Git + Azure DevOps

1. Start with a Workspace
2. Pair the Workspace with a repo
   a) Create a repo in Azure DevOps
   b) Add the repo to a workspace
3. Commit changes to the repo
4. Receive updates from the repo
5. Create feature branches

Outcomes

- Fabric development is backed up in a workspace
- Developers can work on multiple versions of the same item or workspace, quickly replicate the workspace, and merge or roll back their changes

Step 2: Review your architecture

How many workspaces per environment? How many environments?

Varies, general recommendations below for typical data engineering, AI/BI workloads

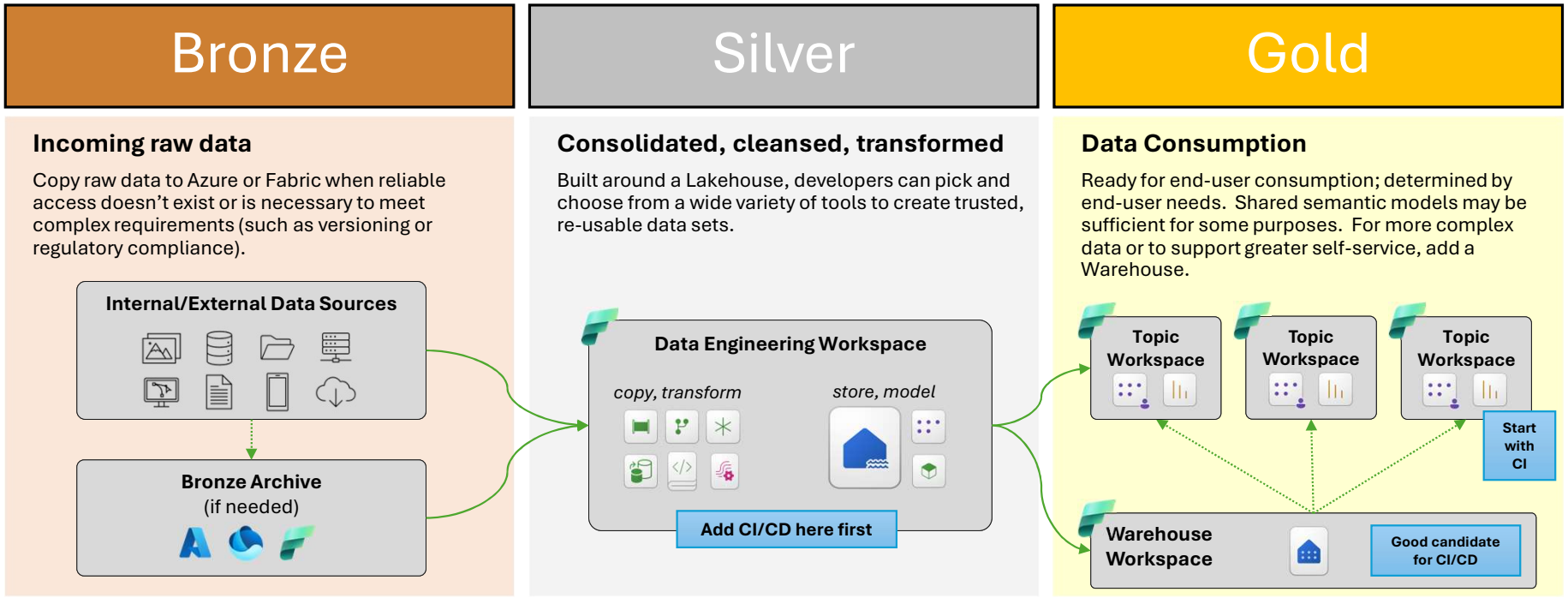
	Data	Code	Content
Typical Items	**Raw, untransformed data** from multiple sources	**Code and config** for transformation, cleansing, feature development, etc.	How **datasets, models, and reports** are presented to the end-users
What to prioritize	**Speed, security, and spend**. Aim for fewer copies and less data movement.	**Re-use and traceability** of both code and data	**Sharing and access**. Organize the content to best serve the audience and manage access.
How many work-spaces?	Keep the source data separate from the code. May copy data to Fabric if necessary but ok to use in place where reliable access exists.	**Single workspace** for transformed data, so developers can blend and re-use cleansed datasets. Allow developers to experiment without impacting existing reports.	**Separate workspaces** are a natural way to group material by topic and manage access. May be organized by topic, department, solution, confidentiality, etc.
CI/CD Needs	Priority is data backup and storage requirements, with **less focus on CI/CD**	These activities are **most likely to benefit** from traditional CI/CD	Self-service and reports created by the business will have **unique CI/CD** needs

Medallion Architectures in Fabric

	Bronze	Silver	Gold
Data	**Raw, untransformed** from multiple sources	**Re-usable:** Consolidated, cleansed, standardized	Ready for **end-user consumption**; modeled with consistent structure
Design Questions	How much control does the organization have and need over the data sources?	Use a single Lakehouse or multiple by project, topic, function?	How do end users need to consume the data?
Tech	**Flexible.** Unlimited data sources. Mirror in Azure storage or Fabric Lakehouse if needed.	Fabric Lakehouse, plus data engineering toolsets	Power BI Semantic Models Fabric Warehouse or Lakehouse
Primary Users	IT and system admin	Data engineers, AI and BI modelers	General business use, including analysts and report developers.

Simplified Medallion Architecture

Optimized for CI/CD, data re-use, and developer collaboration



Step 3: Deployment Pipelines

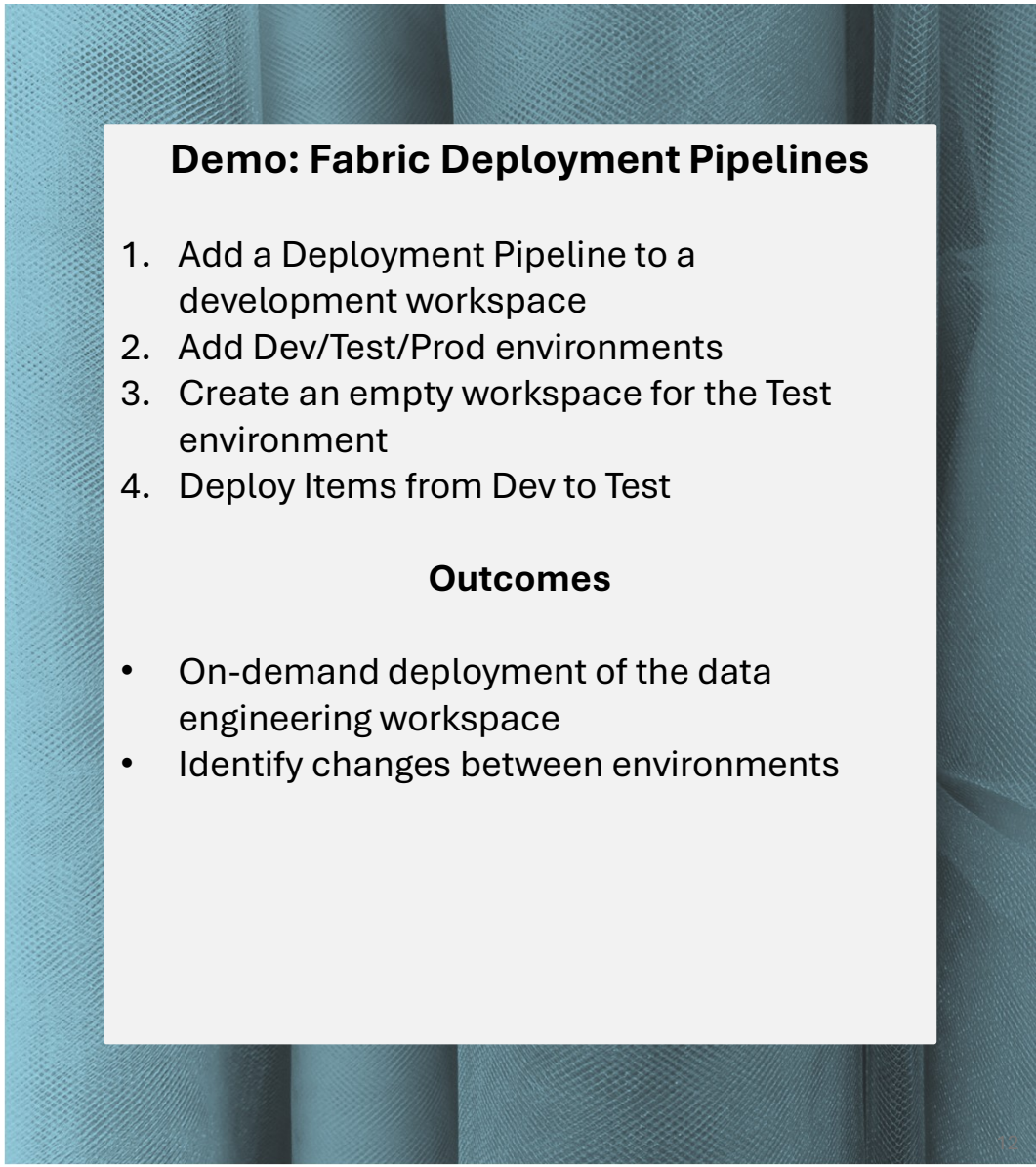
Fabric Deployment Pipelines are used to promote code and configuration from Development → Test → Production, etc.

Advantages

- Simple environment management, included in the Fabric UI.
- Ability to compare items between environments
- Now compatible with nearly all Fabric items
- Ideal for simple and traditional promotion workflows
- Can initiate on-demand or automate via the Fabric REST API

Cautions

- While great for understanding capabilities in Fabric, mature DevOps organizations will turn to dedicated tools.



Demo: Fabric Deployment Pipelines

1. Add a Deployment Pipeline to a development workspace
2. Add Dev/Test/Prod environments
3. Create an empty workspace for the Test environment
4. Deploy Items from Dev to Test

Outcomes

- On-demand deployment of the data engineering workspace
- Identify changes between environments

Challenge:
Data + Connections

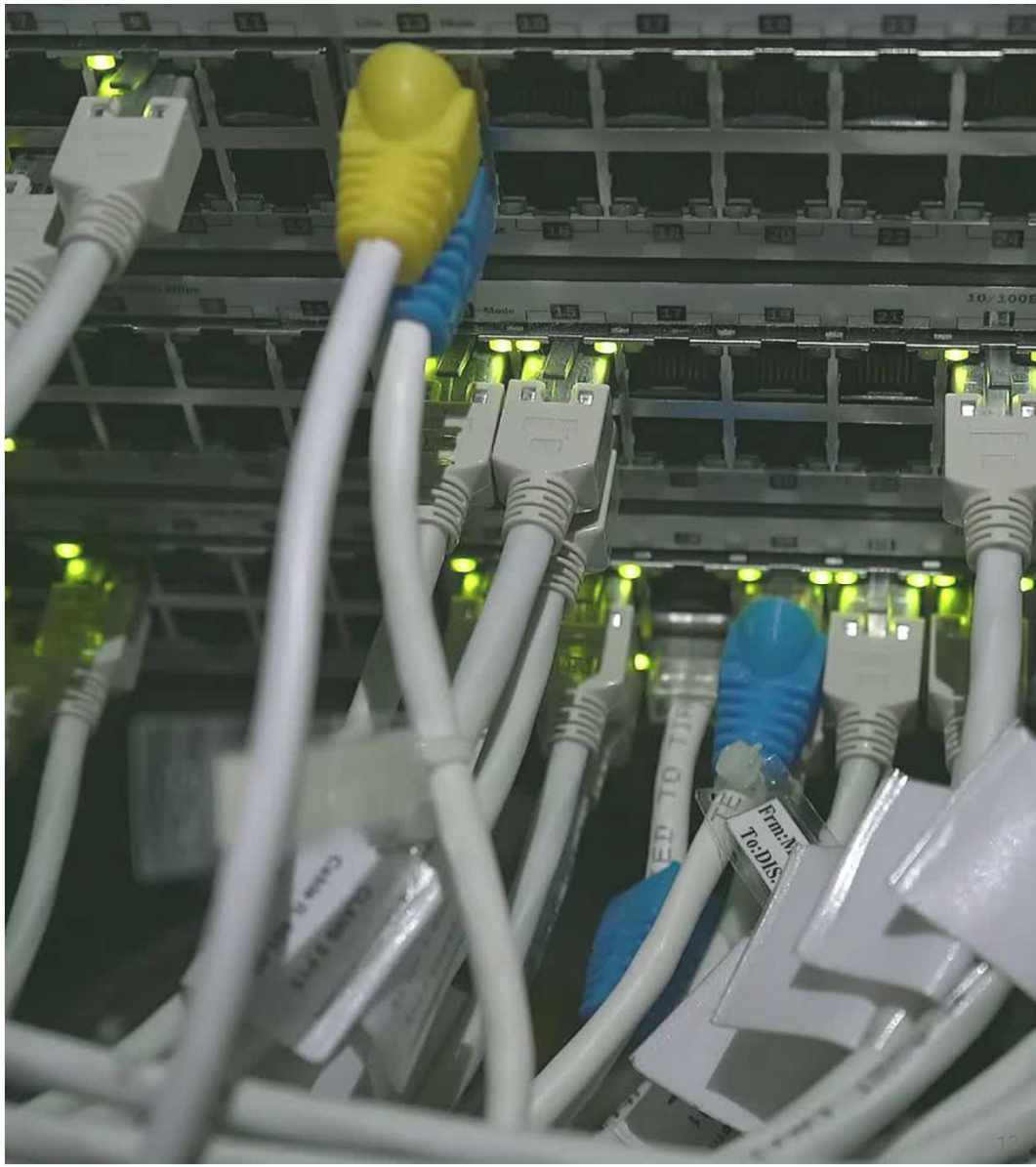
Not included in Deployments

- **Workspace data and files** (i.e. in a lakehouse or warehouse) don't migrate (this is a good thing)

→ Solution: Outside of Bronze, ingest all data via code/config so that it can be easily re-built.

- **Fabric Data Connections**: All the various deployment tools will require some effort to manage data connections when switching environments

→ Solution: Enforce consistency in connections and authorization protocols to reduce errors and improve security



Demo: Fixing Data Connections in Deployment

Scenario	Examples	Fix
Connections that point to an item in the same workspace	Pipelines and notebooks using a Lakehouse in the same workspace	None needed. Each environment workspace will point to content in the current workspace.
Connections that support Dynamic Content or the Variable Library	Pipelines and Notebooks that use the Lakehouse SQL API or reference items outside the current workspace	Add a Variable Library
Power Query M code (Dataflows, PBI Desktop)	Data sources used in Power BI Desktop	Use Parameters. Add deployment rules in the Deployment Pipeline to set.
Connections that don't support Dynamic Content or Parameters	Dataflow Gen 2 Destinations, including Default Destination.	Manually update each item after deployment or automate via scripting.

Advanced Approach: Find/replace all Connection IDs in a Git repo as part of a YAML pipeline. This is especially useful for complex environments with multiple connections, older artifacts, and multiple developers.

The guidelines above are best practice and make it much easier to maintain the environment

Demo: Data and Connections

1. Prepare to hydrate the Lakehouse
2. Review and fix connections
   a) Use a Variable Library
   b) Custom fixes for unsupported items
3. Run pipelines and dataflows

Outcomes

- Test version of the Lakehouse, populated with test data
- Minimal repetitive fixes to data connections when promoting code to test, prod, etc.

YAML Pipelines

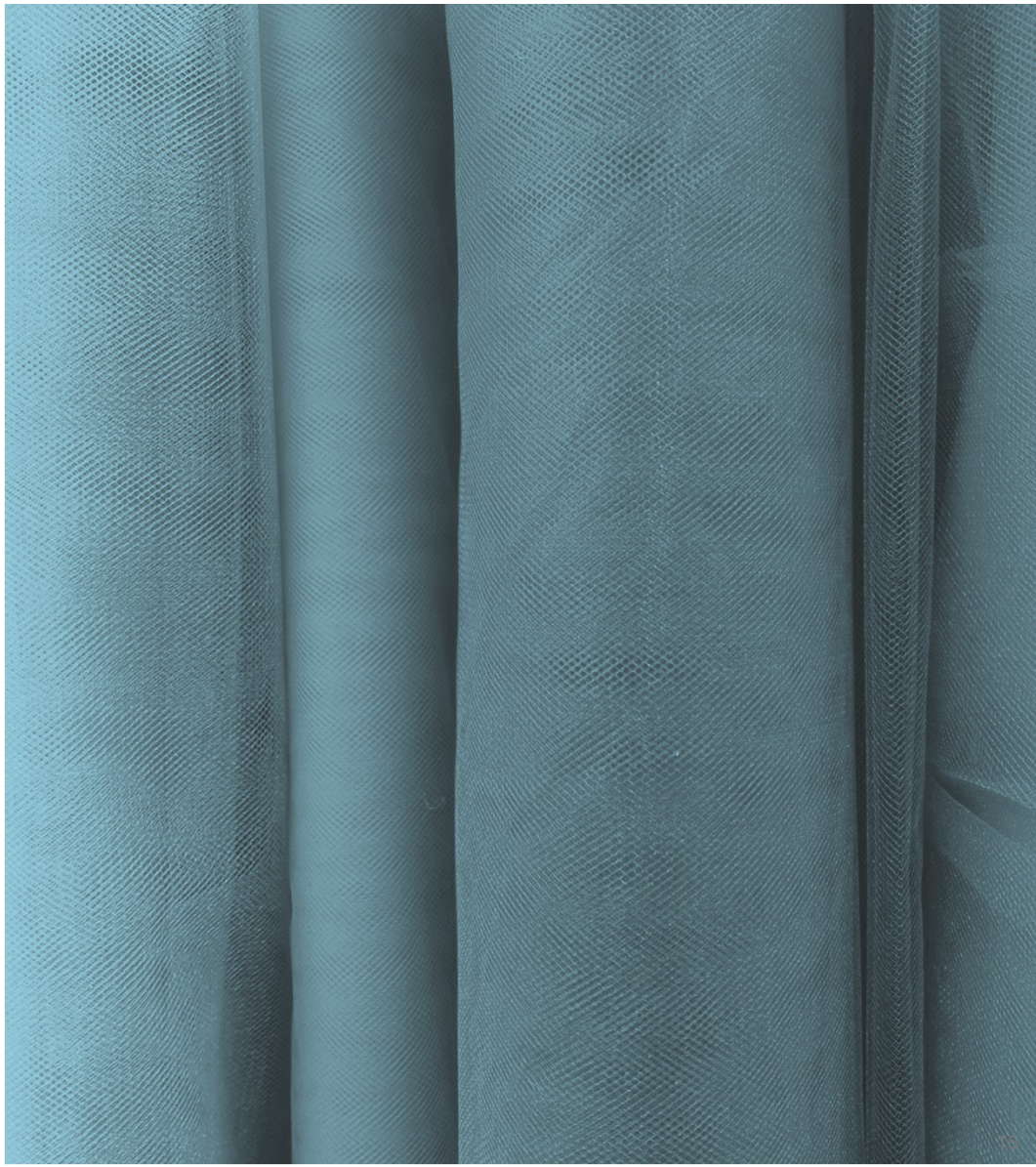
YAML Ain't Markup Language (YAML)

Commonly used for configuration and automation definitions across cloud environments, such as pipeline definitions in Azure DevOps and GitHub.

Customize an example script from the web to get started.

All deployment pipelines rely on the Fabric REST API to access and deploy items in Fabric

Microsoft maintain an open-source Python library, fabric-cicd, meant to be used with YAML configuration, for developers who don’t want to work with REST API directly.



Resources

https://github.com/Left-Join-Fabric/CICD-Resources/ (Notebook to update Dataflow connections)

https://learn.microsoft.com/en-us/fabric/cicd/

https://learn.microsoft.com/en-us/fabric/cicd/manage-deployment

https://learn.microsoft.com/en-us/azure/devops/user-guide/code-with-git

Comfortable with yaml but not wanting to create custom code using the Fabric REST API? Microsoft GitHub fabric-cicd library: https://microsoft.github.io/fabric-cicd/latest/

Questions?

Daniel Schultz

dan@left-join.com

http://www.linkedin.com/in/dschultz101

(LinkedIn QR Code)

Thank you