MICROSOFT FABRIC · CI/CD · PLATFORM ENGINEERING

The Fabric CI/CD Readiness Guide

Five signs your Fabric workspace isn't ready for repeatable deployments — and what to do about each one.

FREE GUIDE · 2025 EDITION

WHAT'S INSIDE

William Weeks-Balconi
Independent Microsoft Fabric Consultant
fabric.acestus.com · jump@hey.com

ACESTUS
FABRIC PRACTICE

INTRODUCTION

Why Fabric CI/CD breaks differently

What makes Microsoft Fabric deployment hard — and why the old ADF playbook doesn't work here.

If you've deployed Azure Data Factory pipelines before, you know the pattern: export ARM templates, run a release pipeline, done. Microsoft Fabric doesn't work that way. The deployment model is fundamentally different, and teams that treat it like ADF end up with broken schedules, drifted workspaces, and no audit trail.

The core problem

Fabric artifacts live in a workspace that has its own state. When you deploy — whether through the Fabric Git integration or a deployment pipeline — artifacts are updated in place. But **schedule triggers, notebook compute settings, and lakehouse connections are not part of the artifact definition.** They live in workspace state.

This means every deployment can silently reset the things your pipelines depend on to run.

What "CI/CD ready" means

A CI/CD-ready Fabric workspace has four properties:

- **Source-controlled:** all artifacts in Git with correct `.platform` files
- **Idempotent:** deploying twice produces the same result
- **Schedule-safe:** deployment doesn't break triggers
- **Auditable:** you can see what changed and roll it back

Most workspaces have none of these on day one. That's normal. This guide helps you assess where you are and what to fix first.

> **THE PATTERN YOU'LL SEE**
>
> A team connects their Fabric workspace to GitHub, commits a few notebooks, runs a deployment pipeline — and their daily refresh that ran at 6am stops running. Nobody notices until the data is stale. The schedule had to be manually re-enabled after every deploy. This is the most common Fabric CI/CD failure pattern, and it's completely avoidable.

How to use this guide

Work through the five problem areas. For each one, you'll get: what the problem looks like, why it happens, and the concrete fix. At the end, score your workspace using the checklist to see where you stand.

> **WHO THIS IS FOR**
>
> Platform engineers setting up Fabric for the first time · Teams already on Fabric who deploy manually · Anyone whose schedules break after a deployment · Organizations migrating from ADF or Synapse to Fabric.

GAP 01 OF 05

Missing or incorrect .platform files

The most common reason Fabric artifacts can't be tracked or deployed from Git.

THE PROBLEM

Fabric can't identify artifacts without .platform metadata

Every Fabric artifact folder needs a `.platform` file that declares the artifact type, display name, and a stable `logicalId` (UUID). Without it, Fabric treats the folder as an unknown file and won't deploy it. With a wrong or duplicated `logicalId`, two artifacts collide in the workspace.

Teams often discover this when they push to Git and the Fabric workspace shows no changes — or worse, when a deployment silently skips half the artifacts.

THE FIX

Generate a new UUID for each artifact's `logicalId`. Use `uuidgen` on Linux or `[System.Guid]::NewGuid()` in PowerShell. Never copy-paste a `logicalId` between artifacts — each must be unique across the entire workspace.

What a correct .platform file looks like

Every artifact folder (`MyNotebook.Notebook/` , `my_pipeline.DataPipeline/` , etc.) must have this file at its root:

EXAMPLE: .PLATFORM FOR A NOTEBOOK

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/fabric/gitIntegration/
platformProperties/2.0.0/schema.json",
  "metadata": {
    "type": "Notebook",
    "displayName": "nb_ing_loanetl_brz"
  },
  "config": {
    "version": "2.0",
    "logicalId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890"
  }
}
```

Valid artifact types

- `Notebook` — PySpark notebook
- `DataPipeline` — orchestration pipeline
- `Lakehouse` — lakehouse artifact
- `Environment` — Spark environment
- `Warehouse` — SQL warehouse

Common mistakes

- Copying a `.platform` file from another artifact (duplicates the `logicalId`)
- Using a placeholder like `00000000-0000-0000-0000-000000000000`
- Wrong `type` value (case-sensitive — `notebook` won't work, must be `Notebook`)
- Missing the `displayName` field (required for workspaces using display-name routing)

> **WATCH OUT**
>
> The Fabric Git sync UI will show "no changes" if a `.platform` file is missing. It won't throw an error. The artifact is simply invisible to the sync engine.

GAP 02 OF 05

Schedules break after every deployment

The most painful Fabric CI/CD failure — and the most overlooked.

THE PROBLEM

Pipeline schedule triggers are workspace state, not artifact state

When you deploy a Fabric Data Pipeline through a deployment pipeline or Git sync, the artifact definition is updated — but the schedule trigger is not part of that definition. It lives in workspace state, and Fabric resets it to "disabled" on deployment.

Your 6am daily refresh stops running. No alert fires. The data goes stale. Someone notices three days later when a dashboard shows last week's numbers.

THE FIX

Build a post-deployment step that re-enables schedules via the Fabric REST API. Save the schedule configuration before deploying and restore it after. Wrap this in a PowerShell or Python script that runs as the last step of your CI/CD pipeline.

The schedule save/restore pattern

Before any deployment that touches a pipeline artifact, capture the schedule:

SAVE SCHEDULE BEFORE DEPLOY

```
GET /v1/workspaces/{workspaceId}/items/{itemId}/jobs/instances?jobType=Schedule
```

Save the response (cron expression, enabled flag, timezone) to a file. After deployment, restore it:

```
POST /v1/workspaces/{workspaceId}/items/{itemId}/jobs/instances
```

Why this isn't documented prominently

Microsoft's deployment pipeline documentation focuses on artifact promotion — moving notebooks and pipelines from dev to prod. The implicit assumption is that you'll configure workspace-level settings (schedules, compute, connections) separately after promotion. In practice, teams don't, and the schedule gets dropped on every deploy.

Detection

After your next deployment, run:

CHECK SCHEDULE STATUS VIA FABRIC REST API

```
GET /v1/workspaces/{workspaceId}/dataPipelines/{pipelineId}
```

Look for `"scheduleConfig": null` or `"isEnabled": false` in the response. If you see either after a deployment, you have this problem.

THIS AFFECTS ALL ARTIFACT TYPES THAT HAVE SCHEDULES

Data Pipelines are the most common case, but Notebooks with scheduled runs and Dataflows Gen2 are also affected. Audit all scheduled artifacts in your workspace.

GAP 03 OF 05

No deployment rules configured

Without deployment rules, every environment gets the same config — including dev credentials in production.

THE PROBLEM

Fabric deployment pipelines pass artifact content as-is without environment substitution

A Fabric notebook hardcoded to connect to `lh_myproject_dev` will point at the dev lakehouse in production unless you configure deployment rules. Most teams discover this when a prod pipeline silently reads dev data — or worse, writes to a dev lakehouse from a production run.

THE FIX

Configure deployment rules in the Fabric portal for each stage transition (dev → test → prod). Use notebook parameter rules to override lakehouse GUIDs per environment. Store environment-specific GUIDs in your `parameters.{env}.json` files under `.deployment/`.

What deployment rules cover

- **Data source rules** — swap connection references per environment
- **Parameter rules** — override notebook parameters (lakehouse IDs, server names, dates)
- **Linked service rules** — point to different Key Vault or storage per env

The .deployment/ folder pattern

Store per-environment overrides alongside your workspace artifacts in Git:

RECOMMENDED FOLDER STRUCTURE

```
ws_myproject/
  .deployment/
    parameters.dev.json
    parameters.tst.json
    parameters.prd.json
```

```
nb_ing_myproject_brz.Notebook/
lh_myproject_brz.Lakehouse/
```

Each `parameters.{env}.json` file declares the environment-specific values that override artifact content at deploy time. These files are the source of truth for "what does dev look like vs. production."

HOW TO VERIFY YOU HAVE THIS SET UP

Open the Fabric deployment pipeline in the portal. Click the arrow between any two stages. If you see "No deployment rules configured," you have this gap. The deployment will succeed but will pass dev config to prod.

GAP 04 OF 05

Hardcoded lakehouse IDs in notebooks

The silent breakage that makes your notebook work perfectly in dev and fail mysteriously in prod.

THE PROBLEM

Lakehouse GUIDs are environment-specific — but notebooks reference them by ID

When a notebook runs `spark.table("lh_myproject_brz.my_table")`, Fabric resolves that against the notebook's default lakehouse attachment — a GUID stored in the notebook's `.platform` metadata. If that GUID points to the dev lakehouse and the notebook is deployed to prod without updating the reference, the notebook silently reads dev data.

Alternatively, teams hardcode GUIDs directly in their PySpark code: `bronze_lakehouse = "lh_myproject_brz"`. This breaks silently when the artifact crosses environments.

THE FIX

Pass lakehouse names as pipeline parameters. Accept them as notebook parameters (`# PARAMETERS CELL`) and read all tables via the parameter: `spark.table(f"{bronze_lakehouse}.{table_name}")` . The deployment rule overrides the parameter value per environment.

The correct notebook parameter pattern

PARAMETERS CELL AT THE TOP OF EVERY NOTEBOOK

```
# PARAMETERS CELL ********************

bronze_lakehouse = "lh_myproject_brz" # overridden by pipeline
silver_lakehouse = "lh_myproject_slv" # overridden by pipeline
etl_process_date = "" # YYYY-MM-DD; blank = latest
```

The parameter defaults make the notebook runnable in dev without any pipeline context. In production, the Fabric pipeline or deployment rule overrides `bronze_lakehouse` to the prod lakehouse name before execution.

Auditing your notebooks

Run this grep across your workspace to find hardcoded lakehouse references:

FIND HARDCODED LAKEHOUSE NAMES

```
grep -rn "spark.table(" ws_myproject/ | grep -v "bronze_lakehouse\|
silver_lakehouse\|gold_lakehouse"
```

Any match that contains a literal lakehouse name (not a variable) is a hardcoded reference that will break across environments.

DEFAULT LAKEHOUSE ATTACHMENT IN .PLATFORM

The notebook-level METADATA block in `notebook-content.py` contains `"default_lakehouse"` and `"default_lakehouse_workspace_id"` fields. These are GUIDs. Deployment rules must update these per environment, or the notebook will resolve table names against the wrong lakehouse.

GAP 05 OF 05

No identity or RBAC plan for CI/CD

Manual deployments run as a person. Automated deployments need an identity — and most teams haven't set one up.

THE PROBLEM

Service principals and managed identities need explicit Fabric workspace access

Teams set up a GitHub Actions workflow or Azure DevOps pipeline, assign a service principal to the subscription, and expect it to deploy Fabric artifacts. It fails. Fabric workspace access is separate from Azure RBAC — you must grant the service principal (or managed identity) Member or Admin access in the Fabric workspace itself.

Additionally, the Fabric REST API requires specific Microsoft Graph scopes. A service principal with only Azure role assignments can't call the Fabric API.

THE FIX

Add the service principal or managed identity as a Member in the Fabric workspace (Settings → Manage access). Grant the Microsoft Graph `Item.ReadWrite.All` and `Workspace.ReadWrite.All` API permissions. For GitHub Actions, use OIDC with a User Assigned Managed Identity (UMI) — no stored secrets.

The recommended identity pattern

RECOMMENDED: UMI + OIDC (NO STORED SECRETS)

1. Create a User Assigned Managed Identity (UMI) in Azure.
2. Add a federated credential: `repo:YourOrg/YourRepo:environment:prd`
3. Add the UMI's service principal as a Member in the Fabric workspace.
4. In GitHub Actions: use `azure/login@v2` with the UMI's client ID.
5. Exchange the Azure token for a Fabric API token via the Power BI token endpoint.

Minimum required permissions

Resource	Permission needed	Where to configure
	Member or Admin	

Fabric workspace		Fabric portal → Workspace settings → Manage access
Azure Key Vault	Key Vault Secrets User	Azure RBAC on the Key Vault resource
Azure Storage / ADLS	Storage Blob Data Contributor	Azure RBAC on the storage account
Microsoft Graph API	Item.ReadWrite.All, Workspace.ReadWrite.All	Azure AD app registration API permissions

COMMON MISTAKE: AZURE CONTRIBUTOR ≠ FABRIC ACCESS

Assigning Contributor on the Azure subscription or resource group does NOT grant access to Fabric workspaces. Fabric has its own access model. You must grant workspace access separately in the Fabric portal.

SELF-ASSESSMENT

The Fabric CI/CD Readiness Checklist

Work through each item. Check the box when your workspace satisfies it. Count your score at the end.

01 · Artifact Structure

- [ ] Every artifact folder has a `.platform` file with a unique `logicalId` — HIGH
- [ ] No two artifacts share the same `logicalId` GUID — HIGH
- [ ] Artifact type values in `.platform` match Fabric's expected casing exactly — MED
- [ ] Lakehouses have all four required files (`.platform` , `lakehouse.metadata.json` , `alm.settings.json` , `shortcuts.metadata.json`) — MED

02 · Schedule Safety

- [ ] Schedules are saved before each deployment and restored after — HIGH
- [ ] Post-deployment verification checks schedule status via Fabric REST API — MED
- [ ] Alert fires if a scheduled pipeline doesn't run within its expected window — LOW

03 · Environment Isolation

- [ ] Deployment rules are configured for every stage transition in the Fabric deployment pipeline — HIGH
- [ ] Per-environment parameter files exist in `.deployment/` in the Git repo — MED
- [ ] No hardcoded dev lakehouse names or GUIDs in production notebook code — HIGH
- [ ] Lakehouse names are passed as notebook parameters, not embedded in code — MED

04 · Source Control

- [ ] All Fabric artifacts are committed to Git (no manually-created-only artifacts in prod) HIGH
- [ ] Git is the source of truth — changes go Git → workspace, never workspace → Git only HIGH
- [ ] PRs are required before changes reach the production workspace MED
- [ ] Branch strategy is documented and followed (e.g., dev → main → release) LOW

05 · Identity & Access

- [ ] CI/CD pipeline uses a service principal or managed identity (not a personal account) HIGH
- [ ] The deployment identity is a Member in the production Fabric workspace HIGH
- [ ] No stored passwords or client secrets — OIDC or managed identity used throughout MED
- [ ] Deployment identity has only the permissions it needs (least privilege) LOW

SCORING

Where does your workspace stand?

Count your checked HIGH-severity items first. They determine your baseline readiness.

HIGH items checked	Readiness level	What it means
0 – 2 of 8	🔴 Not ready	Deployments will fail or silently corrupt your workspace. Fix the HIGH items before anything else.
3 – 5 of 8	🟡 Partially ready	You can deploy, but you have active risk — schedules will break, environments will drift, or your identity setup is fragile.
6 – 7 of 8	🟢 Mostly ready	You're in good shape. A few gaps remain — review the MED items you haven't checked to close them.
8 of 8	✅ Ready	Your workspace has the structural foundations for repeatable CI/CD. Now focus on MEDIUM items for operational hardening.

The most common pattern

Teams typically score 3–4 on HIGH items. The most common unchecked HIGHs are:

- **Schedule restore** — nobody built the post-deploy restore script
- **Deployment rules** — the UI default is "no rules" and it's easy to miss
- **Git as source of truth** — the workspace was configured manually and Git was connected later, so there are artifacts in prod that don't exist in Git

The three fixes that unlock the most value

If you could only do three things, do these in order:

FIX 1 — HIGHEST IMPACT

Build the schedule save/restore script

This single fix eliminates the most common production incident: stale data because schedules went silent after a deployment.

FIX 2

Configure deployment rules for every stage transition

This prevents dev config from leaking into production and is the prerequisite for environment isolation.

FIX 3

Move all pipeline parameters to the PARAMETERS CELL pattern

This makes notebooks environment-agnostic and is the foundation for repeatable, testable deployments.

Want a personalized fix list?

This guide gives you the framework. The review gives you a concrete, prioritized list for your specific workspace — the gaps that matter most for your setup, your team size, and where you are in your Fabric journey.

$50 · Fixed scope · One week

30-min intake call + workspace review + written gap checklist + top 3 fixes.

fabric.acestus.com/contact