## **🚀 What Are Deployment Pipelines in Fabric?**

Deployment pipelines act as Fabric’s built-in **Application Lifecycle Management (ALM)** tool. They help manage the promotion of content (dashboards, dataflows, notebooks, semantic models, SQL databases, etc.) through environments—typically Development → Test → Production. This allows you to:

* Safely collaborate and validate changes before end users see them
* Ensure consistency and version control across environments
* Automate selective or full deployments, with overwrite control based on “item pairing”

## **🛠️ Step-by-Step: How to Create a Deployment Pipeline**

### **Step 1 – Prerequisites**

* Ensure you have a **Microsoft Fabric subscription**.
* You must be an **admin of the Fabric workspace**.
* Each pipeline stage requires a **Premium or PPU-enabled workspace**.

### **Step 2 – Create a Pipeline**

1. Click **Workspaces → Deployment pipelines → Create pipeline** (or **+ New pipeline**).
2. Enter a **name and description**.
3. Define the **stages** (2–10 stages; default: Development, Test, Production). You can rename or adjust as needed.
4. Click **Create (or Create and continue)**.  
    *(Note: you cannot change the structure later.)*

### **Step 3 – Assign Workspaces**

* Assign a separate **workspace** to each pipeline stage. For example, “Dev Workspace” → Development stage, “Test Workspace” → Test stage, and so on.
* If you created the pipeline from inside a workspace, that workspace is pre-assigned.

### **Step 4 – (Optional) Mark Stage Public**

* For the final stage (e.g. Production), you can toggle **"Make this stage public"** so users without pipeline access see it like a normal workspace.

### **Step 5 – Create Content in Development**

* In the Development workspace, create items (reports, semantic models, notebooks, dataflows, lakehouse tables, SQL databases, etc.).
* This content becomes visible in the pipeline canvas for future deployment.

### **Step 6 – Deploy Content to Next Stage**

* Navigate to your pipeline and choose a source stage (e.g. Development).
* Select content to deploy and choose the deployment type:
  + **Full deployment** = all content
  + **Selective deployment** = choose specific items
  + **Backward deployment** = from later stage to earlier (allowed only if target stage's workspace is empty)
* Review changes and add a deployment note if desired.
* Confirm to execute; paired items in target workspace will be overwritten.
* Unpaired items in the target remain untouched.

### **Step 7 – Compare Stages & Review Differences**

* Use the **“Compare stages”** feature to see what changed between stages (e.g. between Dev and Test).
* Changes are highlighted, and for SQL database objects you can view schema-level diffs.

### **Step 8 – Set Up Deployment Rules (Optional)**

* Use **Deployment Rules** to enforce environment-specific settings (such as pointing semantic models to different databases per stage).
* Rules apply during deployment so deployed items inherit stage-specific configuration.
* Note: Not all item types support rules—only supported for dataflows Gen2, semantic models, datamarts, reports, notebooks.

### **Step 9 – Automate with REST API or DevOps**

* You can trigger deployments programmatically using **Fabric deployment pipelines REST API**, or integrate with Azure DevOps / GitHub workflows for CI/CD automation.
* Some teams also adopt tools like **fabric-cicd** to parameterize and manage deployments by branch or environment.

## **📋 Supported Item Types**

Deployment pipelines can move most Fabric assets: **lakehouses, dataflows Gen2, semantic models, reports, dashboards, notebooks**, and **SQL database objects**. Some items (e.g. Data Factory pipelines) are not supported and require manual export or “Save as”.

## **🧩 Visual Summary: Pipeline Workflow**

css

CopyEdit

Development Workspace → Deploy → Test Workspace → Deploy → Production Workspace  
 (workspaces assigned to stages; content flows through)

🚧 Compare changes, apply rules, choose selective or full deployment, document each stage.

## **✅ Key Best Practices**

* Use **separate workspaces for each environment** (Dev/Test/Prod) to avoid confusion.
* Ensure **paired items** align—identical names/type/folders to avoid duplicates.
* Leverage **deployment rules** for stage‑specific configuration where supported.
* For **parameterized or git‑branch-specific deployment**, consider programmatic automation if pipeline UI lacks flexibility.