# Automating Fabric Solution Deployment

This document is designed to help you understand essential concepts with Fabric CI/CD. It will focus on automating Fabric solution deployment using the Fabric REST APIs.

This is the north star. This is the CI/CD story most ISVs want and need.

## The Fabroic

Dddd

A screenshot of a computer

AI-generated content may be incorrect.

## Design Solutions based on Workspaces and Workspace Items

Ddd

A screenshot of a computer screen

AI-generated content may be incorrect.

### Example Scenarios

Here are the four scenarios demonstrated by the FabricSolutionDeployment.

1. Custom Power BI Solution
2. Custom Notebook Solution
3. Custom Shortcut Solution
4. Custom Data Pipeline Solution

#### Custom Power BI Solution

Ss

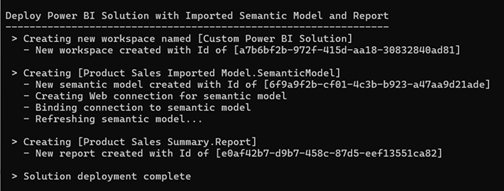
You can also automate the provisioning of a custom Spark environment for executing the code inside notebooks and other types of Spark jobs. Creating a custom Spark environment is valuable if you need to load specific Spark libraries or you need to control the number and size of the nodes in the Spark cluster which processes the execution of code in running notebooks.

You can also leverage workspace items from the **Power BI workload**. For example, you can automate the creation of a workspace followed by the creation of a semantic model which consumes data using import-mode and a Power BI report connected to that semantic model. This example demonstrates how Fabric makes it possible to design and deploy classic Power BI solutions.

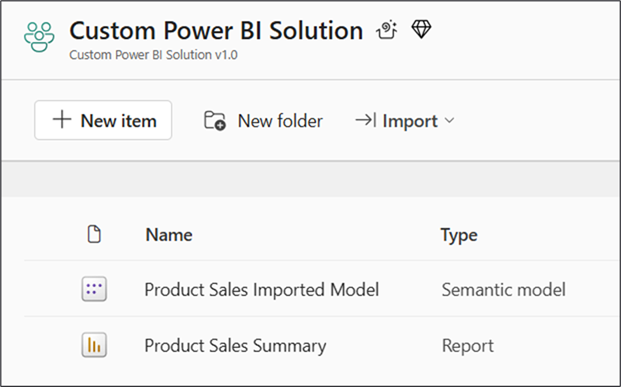
Let’s start with an example of designing a solution with workspace items from the **Data Engineering workload**. Using the Fabric REST APIs, you can create a Fabric workspace with a lakehouse and one or more notebooks. The notebooks can be written to contain Python code which ingests data files and to generates a schema of tables inside the lakehouse. As you will learn, the Fabric REST APIs make it possible to automate running notebooks on demand as part of the solution deployment process.



Sssss



aaa



#### Custom Notebook Solution

Xxxx

Keep in mind you can always mix and match workspace items from multiple Fabric workloads. For example, you can use Data Engineering workspace items to design a solution with a lakehouse and notebooks containing ETL logic used to populate lakehouse tables. Next, you can extend the solution by creating a semantic model in DirectLake mode that consumes data from the table schema of the lakehouse. Then you can complete the solution by creating one or more Power BI reports that consume data from the DirectLake semantic model.

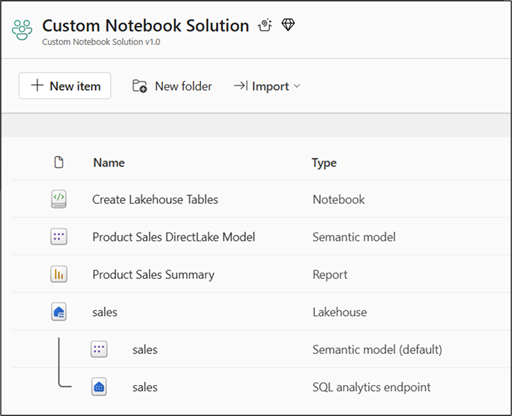


Xxxxx

A computer screen shot of a black screen

AI-generated content may be incorrect.

Xx



#### Custom Shortcut Solution

Xxxx



Xxxxx

A screenshot of a computer screen

AI-generated content may be incorrect.

cc

A screenshot of a computer

AI-generated content may be incorrect.

xxx

#### Custom Data Pipeline Solution

Sssssss

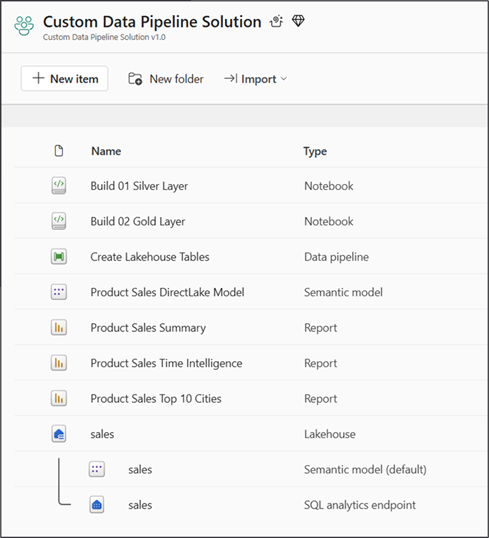


It usually makes sense to implement the PoC by hand at first using Fabric’s browser-based UI experience. You can start by creating a workspace. After that, you can create and configure a set of workspace items from whichever Fabric workloads you need. Once you have implemented the PoC by hand, you can test out your solution to verify that it scales as required and that it behaves the way you expect it to.

A screenshot of a computer screen

AI-generated content may be incorrect.

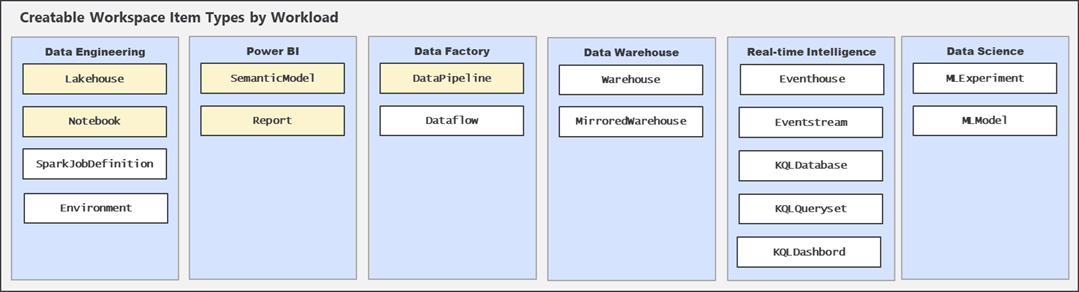
Aaaaa



x

As you begin developing solutions for Microsoft Fabric, you should design your solutions in terms of workspaces and workspace items. As you learn more about the different Fabric workloads and the types of workspace items they offer, you will become more experienced in architecting end-to-end solutions.

Fabric solutions designed and implemented in terms of workspace items. Developers can discover, create and manage workspace items inside scope of a workspace. Workspace items are created and updated using item definitions

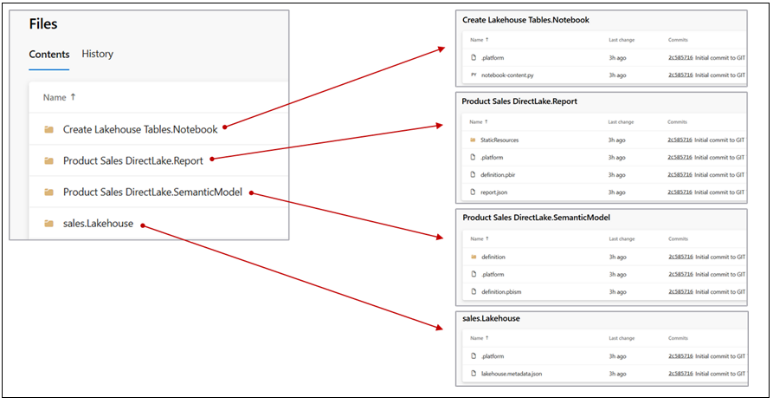


You need to know the details of all the workspace item types you have in a solution. This proof of concept deals with the following workspace items

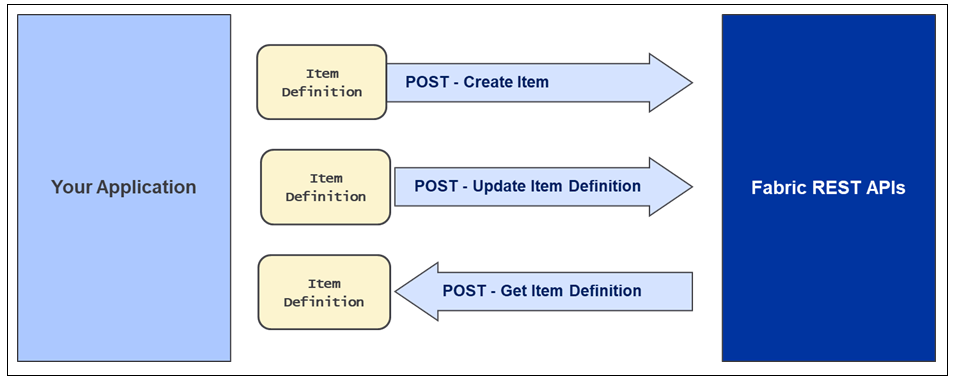
1. Lakehouses
2. Lakehouse shortcuts
3. Notebooks
4. Data pipelines
5. Semantic models (Import and DirectLake)
6. Reports

Item Definitions

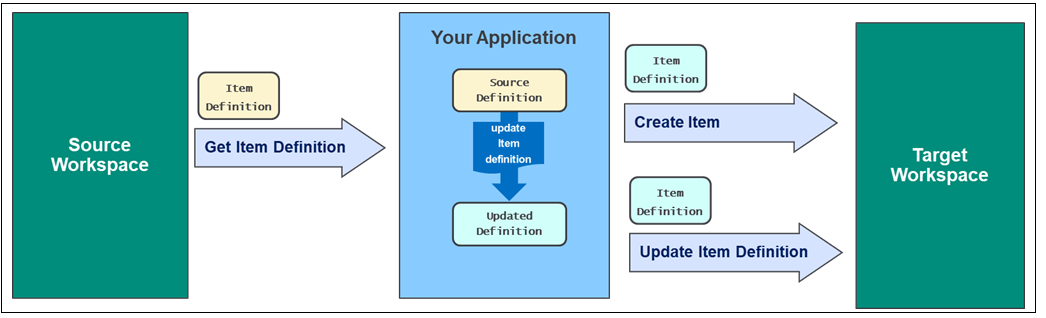
Fabric ALM strategy with CI/CD based on item definition files. Fabric automatically serializes workspace items into set of item definition files. Each type of workspace item defines the set of files that constitutes a valid item definition



Fabric items can be created and updated using item definitions. You can pass item definition when calling Create Item API. You can modify existing workspace item by calling Update Item Definition passing item definition. You can retrieve item definition for existing workspace item by calling Get Item Definition.



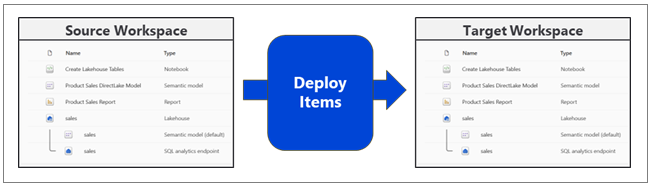
Fabric items can be cloned using item definitions. Retrieve item definition from source workspace by calling Get Item Definition. Manipulate content of item definition part files such as substitute source Ids for target Ids. Pass manipulated item definition to Create Item API to create new item. Pass manipulated item definition to Update Item Definition API to update existing item.



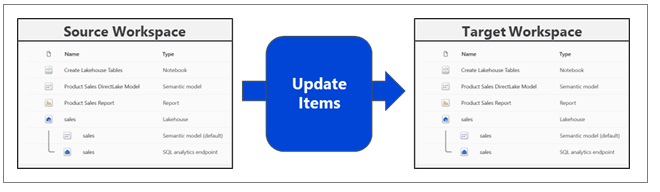
Here’s some good news. Once you implement a Fabric solution by hand, you can reverse engineer item definitions from the existing workspace items. You accomplish this calling the **Get Item Definition** API and storing the response as a set of item definition files. This technique will allow you to acquire the resources you need to generate item defintions you can use to call the **Create Item** API and the **Update Item Definition** API. This guidance document will revisit this essential topic in the **Create and Update Workspace Items** chapter.

## Developing Custom API-driven Pipelines

Deploy workflow creates new target workspace and clones source workspace items. Deploy workflow always creates target workspace and builds it from scratch. Fabric REST APIs provide necessary CRUD APIs for managing workspace items.



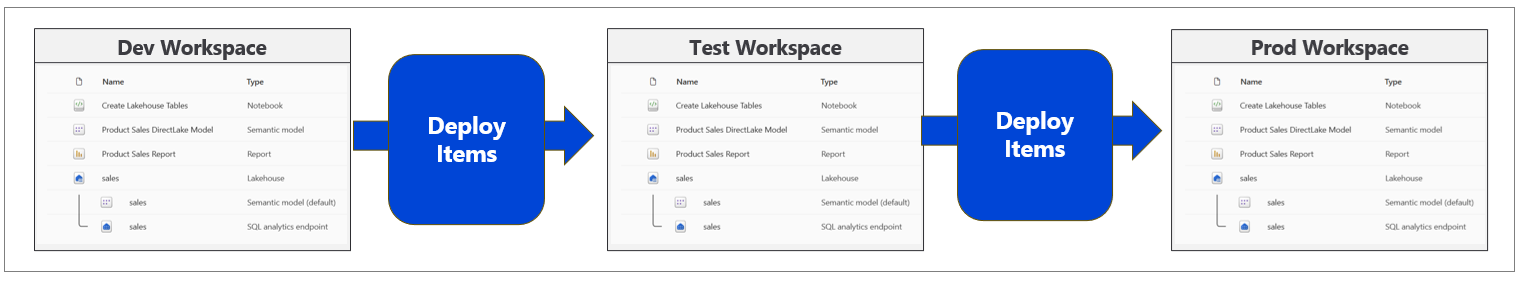
Update workflow processes set of updates on target workspace that already exists. Source workspace items that already exist in target workspace are updated. Source workspace items that do not already exist in target workspace are created.



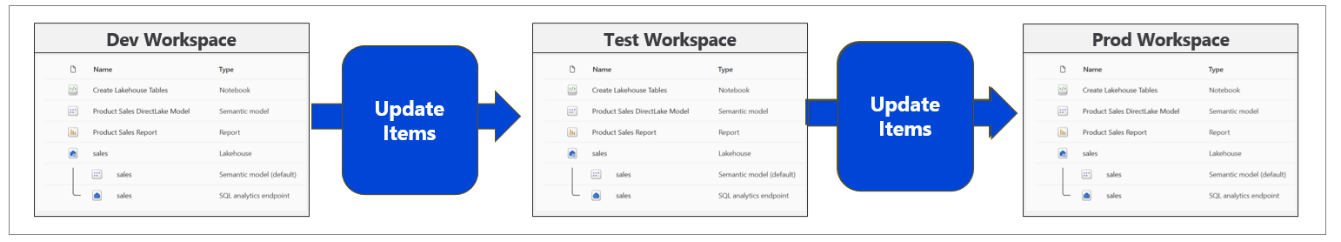
### Motivation for Staged Deployments

Staged deployment used to push changes between environments like **DEV**>**TEST** >**PROD**

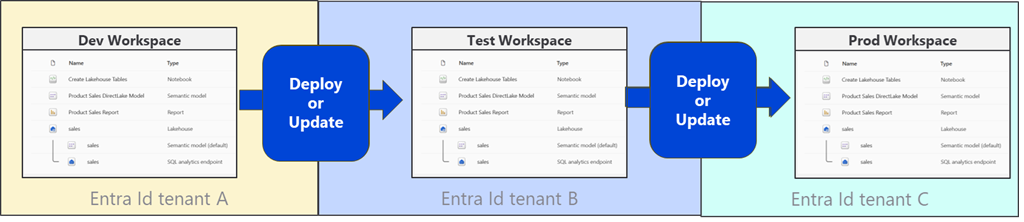
**DEPLOY** workflow creates downstream workspaces on demand



**UPDATE** workflow pushes out full update to all workspace items or partial update



DEPLOY and UPDATE workflows can cross environments such as Entra Id tenant boundaries



Building each stage requires configuring different datasource paths

DEPLOY workflow for must configure datasources differently for DEV, TEST and PROD

### Motivation for Multitenancy

**Multitenant application development** is a software architecture that allows an ISV to serve multiple customers using a single instance of an application. In a multitenant architecture, each customer is considered to be a separate **tenant.** You can think of an analogy with a large apartment building where each tenant has their own apartment. A requirement of multitenancy is that each tenant is created in isolation from all other tenants.

If you have worked with Entra Id (formerly Azure AD), the word **"tenant"** might make you think of an Entra Id tenant. However, the concept of a tenant is different when designing a multitenant application for Fabric. In this context, each tenant represents a customer with one or more users. With the proper planning, you can build a multitenant environment with Fabric which scales to 100s or 1000’s of customer tenants scope inside a single Entra Id tenant.

When developing multitenant applications for Fabric, it’s a best practice to create a separate workspace for each customer tenant. By provisioning each customer tenant using a separate workspace, you can provide a base level of isolation. In a more complicated solution design, it might make sense to create multiple workspaces for each customer tenant. However, a design based on a single workspace per customer tenant is a good place to start.

Fabric provides development platform for building multi-tenant applications. Each customer tenant created using one or more Fabric workspaces.

Developer uses Fabric REST APIs to create and configure workspaces and workspace items

A diagram of a workflow

Description automatically generated

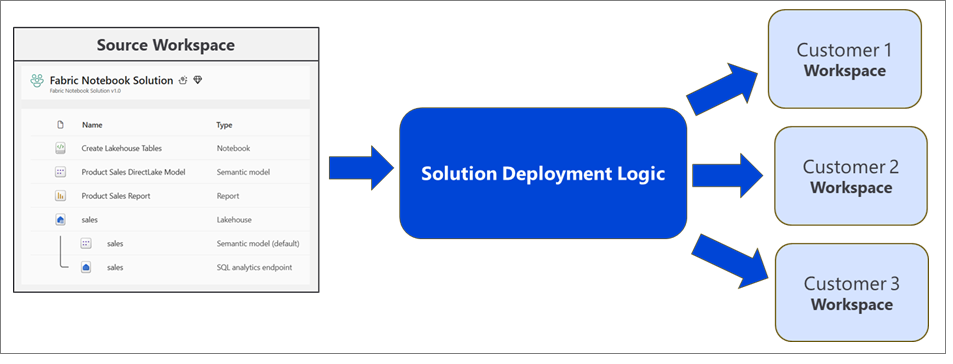
When developing a multitenant application, it’s essential that you learn to fully automate the process of provisioning new customer tenants. This provisioning process typically involves creating a new workspace and then creating and configuring a set of workspace items inside. If parts of the tenant provisioning process require manual intervention, that can limit your ability to scale up to a large number of customer tenants.

Design workspace to serve as solution template to provision customer tenants

Implement DEPLOY workflow to create target workspaces from source workspace

Execute deployment workflow once to create each customer tenant

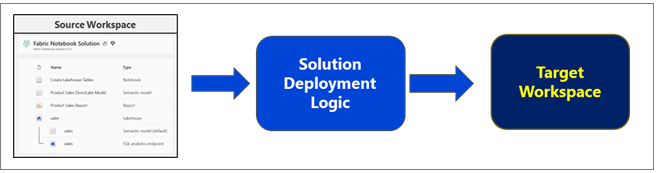
DEPLOY workflow must support parameterization of customer data such as datasource paths



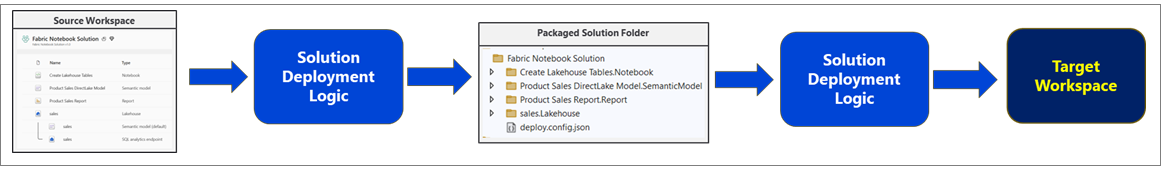
Preview what we will be using from Fabric REST APIs.

### Deployment Options

Use a live workspace as the source solution template



Use Fabric item definitions exported to packaged solution folder (requires deploy.config.json)



### Mixing with GIT Integration

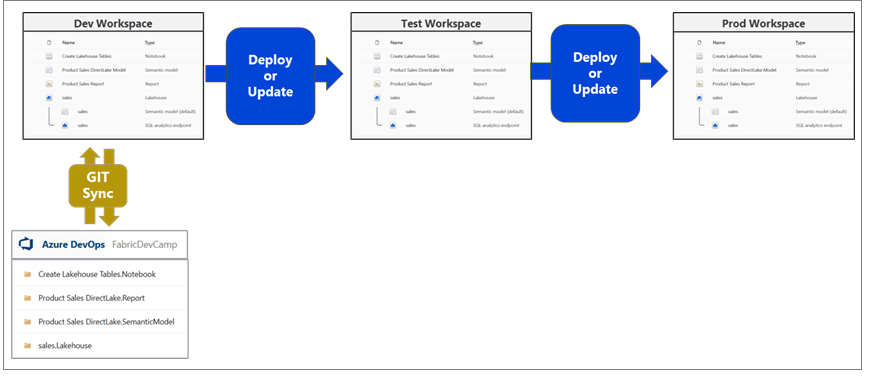
High-level architecture for CI/CD on a single target workspace

Changes are propagated to DEV workspace using feature workspaces and pull requests

Changes pushed from DEV to TEST to PROD during testing and approval process

Create customer tenants by running DEPLOY workflow from PROD

Update customer tenants by running UPDATE workflow from PROD



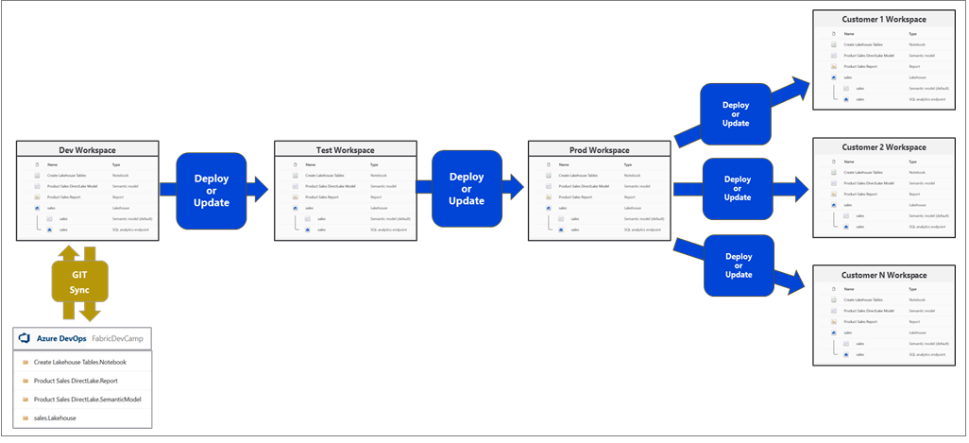
High-level architecture for CI/CD with multiple target workspaces

Changes are propagated to DEV workspace using feature workspaces and pull requests

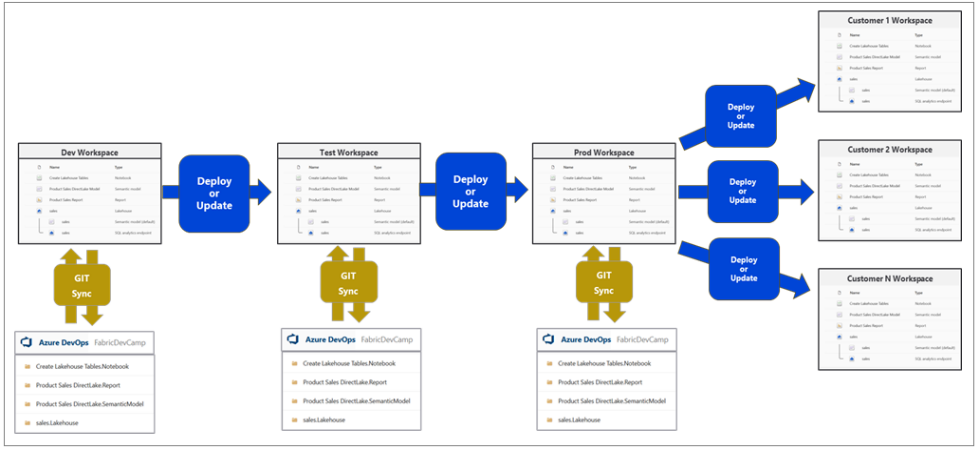
Changes pushed from DEV to TEST to PROD during testing and approval process

Create customer tenants by running DEPLOY workflow from PROD

Update customer tenants by running UPDATE workflow from PROD



Sss



### Understanding Workspace Item Dependencies

Create workspaces using the **Create Workspace** API.

Create workspace items using item definitions and the **Create Item** API.

You will also see using the job scheduler to run a notebook using the **Run On Demand Item Job**

Create connection using the **Create Connections** API. .

Retrieve item definitions using the **Get Item Definition** API.

Update the definition for existing workspace items using the **Update Item Definition** API.

API.

Here are the demonstration deployment workflows

* Demo 1 - Deploy Fabric Solutions using Item Definitions
* Demo 2 - Deploy Fabric Solutions using Workspace Template
* Demo 3 – Deploy Fabric Solutions using Item Definition Files in Source Control

Demo 1 - Deploy solutions using item definitions

***graphic***

Demo 2 - Deploy solutions using Workspace Template

***graphic***

Demo 3 – Deploy solutions using Item Definitions from Source Control

***graphic***

## Challenges in Fabric Solution Deployment

Yada yada

### Manage Workspace Item Dependencies

With a Fabric workspace, workspace items will often have dependencies on other workspace items.

Many workspace items have dependencies on other workspace items

Notebook definition depends on workspace id and lakehouse id to reference its default lakehouse

DirectLake semantic model definition depends on connection string to SQL endpoint of lakehouse

Report definition depends on semantic model id

After deployment, solution should not have dependencies on other workspaces

All dependencies should be self-contained in new workspace

Fabric solution deployment requires custom logic to update dependencies accordingly

A screenshot of a computer

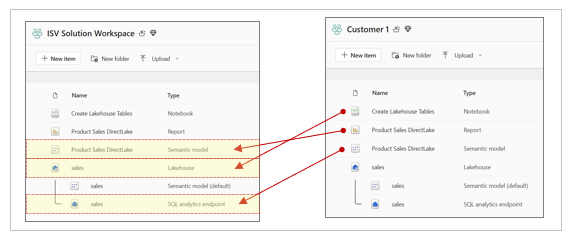
AI-generated content may be incorrect.

When deploying a solutions, you must understand these dependencies.

* **Lakehouse** has no dependencies on other workspace items
* **Notebook** has dependencies on workspace id & lakehouse id of default lakehouse
* **DirectLake semantic model** depends on SQL endpoint connect string and database Id
* **DirectLake semantic model** might depend on running a notebook to create lakehouse tables
* **Report** depends on id of semantic model to which it is bound

Deployment workflow must create workspace items with dependencies last. Notebook definition depends on lakehouse id for its default lakehouse. DirectLake semantic model definition depends on connection string to SQL endpoint of lakehouse. Report definition depends on semantic model id

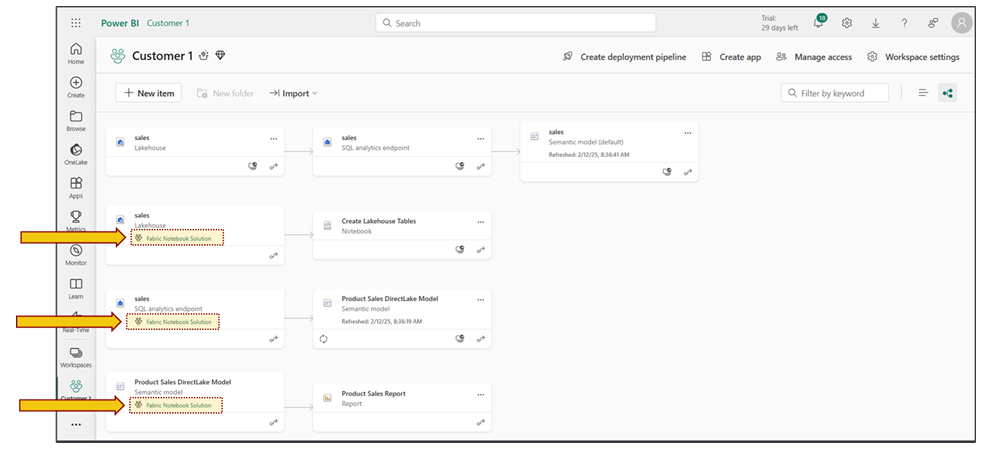
Shallow copy of template workspace is not the result you want. Workspace for customer tenant should not have dependencies on source workspace



Checking for intra-workspace dependencies

Navigate to Lineage view to see item dependencies

Look for items that have dependencies on items in other workspaces



Item

For this reason, a Fabric solution must deploy its workspace items in this order

* Lakehouses
* Notebooks
* Semantic models
* Reports

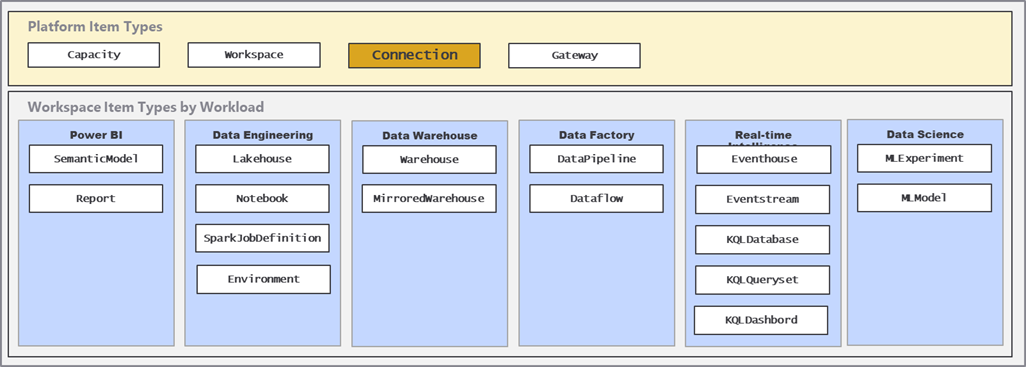
And now a little more detail about the workflow

* Create lakehouse
  + Track lakehouse id for later use when updating notebook dependency
  + Track lakehouse properties which provide connection information for SQL endpoint
  + Track lakehouse name to later determine which semantic models are default for lakehouse
* Create notebooks
  + Create notebook using item definition which is updated to include workspace id & lakehouse id
  + Run notebook and monitor execution until completion to ensure lakehouse tables are created
* Create DirectLake semantic models
  + Create semantic model using updated item definition that includes SQL endpoint connect string
  + Track semantic model id for later use when binding report
  + Create SQL connection to lakehouse SQL endpoint and bind it to semantic model
  + Refresh semantic model
* Create Power BI reports
  + Create report using updated item definition that includes semantic model id

Summary of section

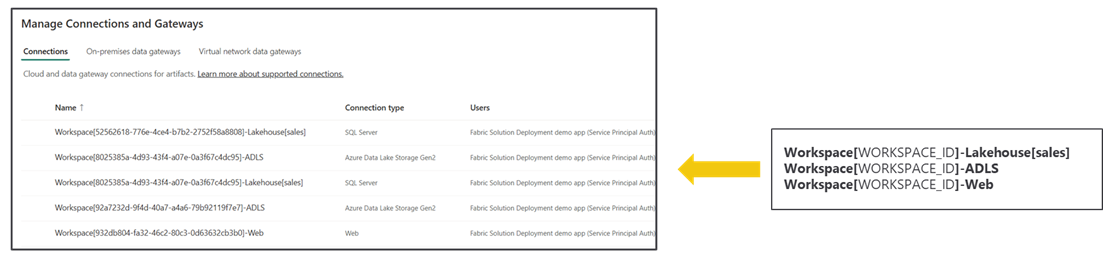
### Challenge of Managing Connections at Workspace Scope

Unlike workspace items, connection items are scoped at level of Entra Id tenant. No API support for workspace-level management of connections. No support in Fabric GIT integration to track, backup, restore, recreate connections. Developer must find innovative approach for managing connections at scope of workspace.



### Using a Naing Convention for Managing Connections

Using connection naming convention allows for management at workspace scope. Connection name contains Workspace Id. Connection name also indicates connection type (e.g. Lakehouse(sales) vs ADLS vs Web).



Benefits to connection naming convention

* It allows for cascading connection deletes when deleting workspace
* Allows for discovery of connection when inspecting source workspace
* Enables configuration when recreating lakehouse shortcuts and data pipelines

### Parameterizing Datasource Paths and Customer Data

Where Do Datasource Paths Live in a Solution?

Inside item definition files in semantic models. Example item definitions files with datasource paths include expressions.tmdl and model.bim.

Inside Notebooks. Developers can add URLs directly in their python code

In connections. A connection is a credential bound to a datasource path

In shortcut. Shortcut created with datasource path

Shortcut must be bound to connection with same datasource path

In data pipeline. Data pipeline created to reference connection to external datasource

Connection created with target datasource path

## Create and Update Workspace Items using Item Definitions

The Fabric REST API programming model introduces the abstraction of the ***item definition*** which used as a mechanism to create and update workspace items. At a high level, an item definition represents a set of system files make up the definition of a workspace item. Each of the system files in an item definition is known as a **part**. While all item definitions are constructed using a common format, each workspace item type defines its own set of parts required to fill out a complete definition.

There are three primary scenarios in which you will program directly with item definitions. First, you can pass an item definition when calling the **Create Item** API. Second, you can retrieve an item definition for existing workspace item by calling the **Get Item Definition** API. Third, you can modify an existing workspace item by passing an item definition when calling the **Update Item Definition** API.

A diagram of a process

Description automatically generated

When calling the **Create Item** API, the item definition is passed in the POST request body as part of the JSON payload. To create a notebook, you can call the **Create Item** API passing an item definition which contains a single system file part name **notebook-content.py**. As shown in the following diagram, an item definition contains a **parts** property collection which contains a set of one or more **part** files.

A screenshot of a computer

Description automatically generated

You should keep in mind that item definitions must be represented in a JSON structure that can be passed across the network in API calls. This leads to an important question. How can you embed the content of a file inside a valid JSON structure? The answer is encode the file contents of item definition parts using Base64 encoding.

In order to create an item definition, you must first convert the contents of each item definition part file into a Base64 encoded format. After that, you can add the encoded file contents into the **payload** property of a part as an ordinary string as shown in the following JSON code listing.

{

"displayName": "notebook1",

"type": "Notebook",

"definition": {

"parts": [

{

"path": "notebook-content.py",

"payload": "{PY\_FILE\_CONTENT\_BASE64\_ENCODED}",

"payloadType": "InlineBase64"

}

]

}

}

Note that each part in an item definition requires three properties which are **path**, **payload** and **payloadType**. Each **part** is added in a **parts** collection which is a property of the **definition**. You have seen that the item definition for a notebook is fairly simple in that it only includes a single file. However, the item definitions for other workspace item types often contain multiple files.

Here’s what can be a bit confusing at first. Some workspace item types support creation and updates using item definitions while others do not. For example, you will use definitions to create some types of workspace items such as notebooks, Spark job definitions, semantic models and reports. However, you will not use an item definition when creating other types of workspace items such as lakehouses and warehouses.

Keep in mind the details and programming techniques for creating workspace items will vary from one type of workspace items to another. Some workspace item types do not support creation or update using items definitions while other types of workspace items require it. Each type of workspace item defines its own unique set of definition files required and allowed in the parts collection of an item definition. For example, the item definition for a notebook requires a Parts collection with a single file named **notebook-content.py** which contains the code for the notebook. The item definition for a semantic model or a report requires are parts collection with multiple definition files. The number of files in the parts collection for these types of item definitions can number into the 100s.

A keep point is that each new type of workspace items brings along its own degree of complexity. In this article, we will examine a scenario with just four workspace items types which are lakehouses, notebooks, semantic models and reports. This article and the accompanying **FabricCICD** sample project have limited the scope to these four types to create a complete end-to-end workflow for solution deployment.

However, you will be required to deal with other types of workspace items that are not covered here.

## Deploy Fabric Solutions using Item Definitions

## Deploy Fabric Solutions using Workspace Template

## Deploy Fabric Solutions using Item Definition Files in Source Control