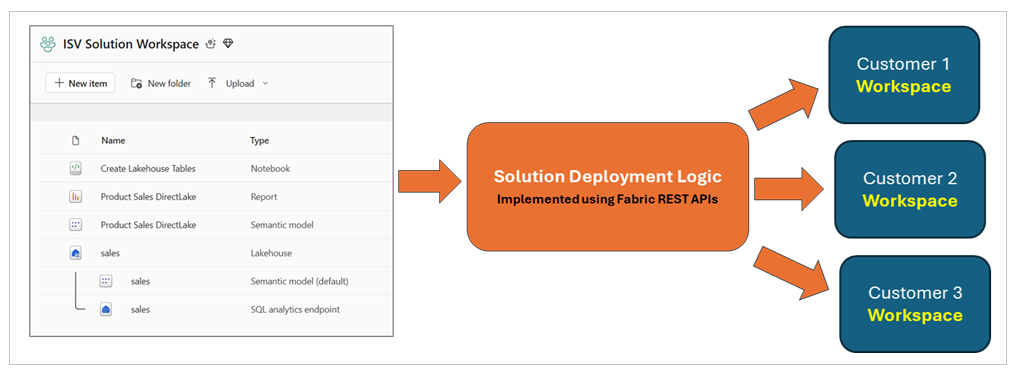
# Automating Fabric Solution Deployment

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.



Preview what we will be using from Fabric REST APIs.

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***

## Manage Workspace Item Dependencies

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

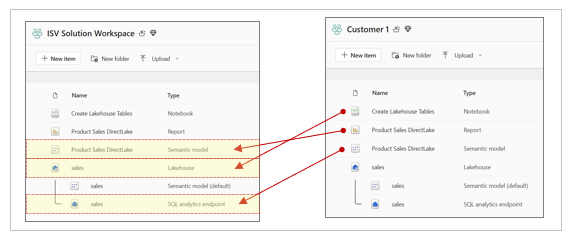
A screenshot of a computer

AI-generated content may be incorrect.

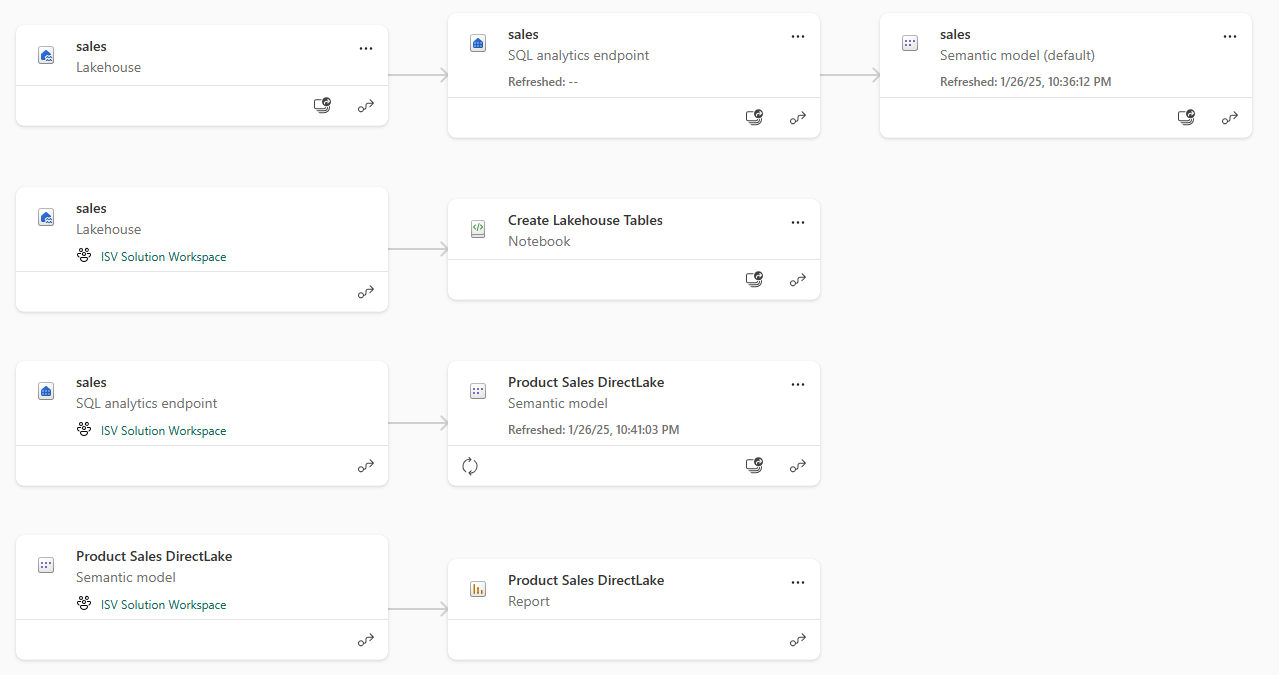
When deploying a solutions, you must understand these dependencies.

* **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

When cloning a



xxx



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

## 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