# Chapter 01 - Introduction to the Fabric REST APIs

Microsoft Fabric is a platform that provides the ability to design, implement and deploy data-centric solutions. Fabric makes it possible to build end-to-end solutions by offering services for data movement, processing, ingestion, transformation, real-time event routing, semantic modeling and report building. Fabric services are exposed through a set of **Fabric** **workloads** which include Data Factory, Data Engineering, Data Warehouse, Data Science, Real-Time Analytics, Power BI and Data Activator.

A blue logo with green text

Description automatically generated

Microsoft promotes Fabric as a SaaS platform that simplifies life for users and administrators alike. Fabric has also been designed to offer a full-fledged development platform to professional developers and independent software venders (ISVs). The Fabric REST APIs extend the Fabric platform by giving developers the ability to automate the deployment and management of Fabric solutions in a production environment. Fabric also provides ALM and CI/CD features which leverage the platform’s GIT integration to provide organizations with a robust and reliable way to update solutions after they have been deployed into production.

## Design Solutions based on Workspaces and Workspace Items

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.

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.

A close-up of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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.

A close-up of a computer

Description automatically generated

What if you need to migrate a data-centric solution from another data platform that is based on SQL. That’s where the **Data Warehouse workload** comes in by allowing you to design SQL-based solutions using a warehouse which is created as a workspace item. You can then extend a Data Warehouse solution with complementary workspace items such as a lakehouse for staging data and data pipelines to ingest data files and to execute SQL statements against the warehouse’s SQL endpoint to create and populate tables in the warehouse.

A close-up of a logo

Description automatically generated

If you need to design a solution which can handle a large volume of incoming data, you can leverage workspace items offered by the **Real-time Intelligence workload**. You can design a Real-time Intelligence solution by creating an eventhouse with an eventstream configured to write its output into a KQL database. You can further extend the solution by adding KQL querysets and KQL dashboards.

A screenshot of a computer

Description automatically generated

Once you’ve designed a solution using workspace items from the Real-time Intelligence workload, you can further extend it with a Reflex item from the **Data Activator workload**. This makes it possible to create reactionary triggers which fire to initiate some type of action in response to certain conditions being met as data is ingested.

What’s the key takeaway? Workspace items are the fundemtnal building blocks used to design, implement and deploy Fabric solutions.

Once you’ve designed a Fabric solution using a composition of workspace items, you can move forward to test out your theory by implementing the solution in a Proof of Concept (PoC). The PoC is important because it allows you to verify that the solution scales to the anticipated volume of data and that the solution does what it’s supposed to.

Over the next few chapters you will learn how to automate the creation of workspaces and workspace items using the Fabric REST APIs. As you will soon learn, workspace items are created using the **Create Item** API together with a special type of creatable item template known as an **item definition**. Learning how to program with item definitions will be one of the most essential skills you need to learn as a Fabric platform developer.

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.

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.

## Develop Multitenant Applications on the Fabric Platform

**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.

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.

A critical aspect of designing multitenant applications has to do with achieving the required level of isolation. The last thing you want is for a user from one customer tenant to see data from another customer tenant. Creating each new customer tenant in its own Fabric workspace provides the first level of isolation.

A second level of isolation can be achieved by using a different service principal to create and populate each customer workspace. A multitenant design which requires a one-to-one mapping between customer tenants and service principals prevents any Entra Id identity from having access to more than just a single tenant.

## Fabric as a PaaS Platform for Developers

Microsoft actively promotes Fabric as a **Software as a Service (SaaS)** platform. However, Fabric can also been seen as a powerful **Platform as a Service (PaaS)** platform from the perspective of an ISV. For example, an ISV can build a multitenant application that leverage Fabric’s PaaS features while at the same time making their own application available to their customer base as a SaaS offering.

A close-up of a square

Description automatically generated

By leveraging Fabric as a PaaS platform, an ISV can extend their reach to users and organizations that do not use Entra Id. An ISV can authenticate the users of their SaaS application using whatever identity provider they’d like. For example, some ISVs may prefer to authenticate their users with an identity provider other than Entra Id such as Okta, Auth0 or OneLogin. Once the ISV application authenticate the user, it can validate the user’s identity and enforce access control policies. However, the ISV application does not propagate the user’s identity to Fabric when it calls the Fabric REST APIs. Instead, the ISV uses a service principal to call the Fabric REST APIs.

## Fabric REST API Architecture

If you’re an experienced software developer, it’s likely that you’ve learned to work with quite a few different APIs before. If you’ve programmed with other Microsoft APIs such as the Microsoft Graph API or the Power BI REST API, you will find that programming with the Fabric REST API will be familiar.

The Fabric REST APIs are designed to be accessible to any developer on any development platform. To this end, Fabric REST API architecture has been designed using the principles of REST and open security standards which include OAuth2 and Open ID Connect. You can develop a custom application that executes API calls by submitting HTTP requests against Fabric REST API endpoints.

A close-up of a computer screen

Description automatically generated

If you are working within the Microsoft public cloud, Fabric REST API endpoints are accessible through the base URL of **https://api.fabric.microsoft.com/v1**. If you are working in a different cloud such as a sovereign cloud or a government cloud, the base URL will be slightly different.

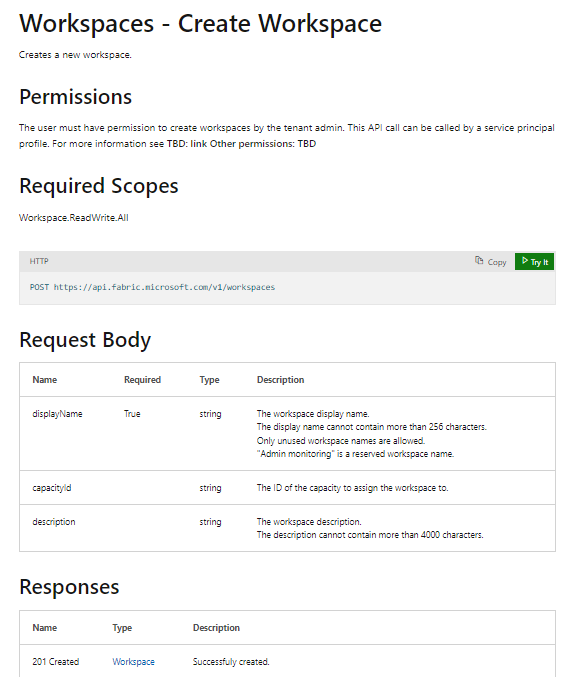
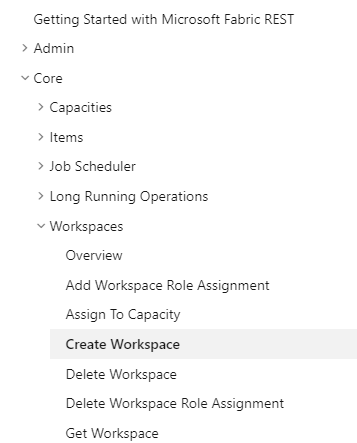
The Fabric REST API is secured by the Entra Id Service. Before calling the Fabric REST APIs, your application must first acquire an access token from the Entra Id Service by executing an authentication flow. After that, your application must transmit the access token each time it calls to the Fabric REST APIs. This is accomplished by passing the access token in the **Authorization** request header.

## Fabric REST API Documentation

If you plan on developing solutions with Fabric, you should become familiar the Fabric REST API documentation. This is an essential developer resource which is available at the following URL.

* <https://learn.microsoft.com/rest/api/fabric>

The Fabric REST API documentation provides essential details for developers such as the requirements for constructing REST URLs and structuring the JSON that goes into a request body. This documentation also tells you what you can expect in the response in terms of HTTP status codes and the structure of JSON in the response body.



## The Fabric ISV Playbook Developer Sample

This guidance document is accompanied by a developer sample project named **FabricIsvPlaybook**. The code in this project demonstrates how to deploy Fabric solutions using the Fabric REST APIs by creating and configuring workspace items. The **FabricIsvPlaybook** project is a simple .NET console application built using .NET 8, C# and the Fabric REST API .NET SDK. Source code for the **FabricIsvPlaybook** project is available in a public GitHub repository at the following URL.

* <https://github.com/PowerBiDevCamp/Fabric-ISV-Playbook>

You can also download all the project files for **FabricIsvPlaybook** project as a single ZIP archive using [**this link**](https://github.com/PowerBiDevCamp/Fabric-ISV-Playbook/archive/refs/heads/main.zip).

Once you have downloaded the **FabricIsvPlaybook** project source code to a local folder, you can open it using Visual Studio 2022. The high-level project structure of the **FabricIsvPlaybook** is shown in the following screenshot.

A screenshot of a computer

Description automatically generated

You can open the **FabricIsvPlaybook** project using any version of Visual Studio 2022 including the community version which is free. If you need to download a version of Visual Studio 2022, you can visit [**this link**](https://visualstudio.microsoft.com/downloads/).

In the root folder of the **FabricIsvPlaybook** project, there are source files named **Program.cs** and **AppSettings.cs**. These are the two primary source files you will update as you experiment with this developer sample. You need to edit the values for configuration settings in **AppSettings.cs** for various demos to work correctly. You will also be commenting and uncommenting lines of code in **Program.cs** to switch between different demos.

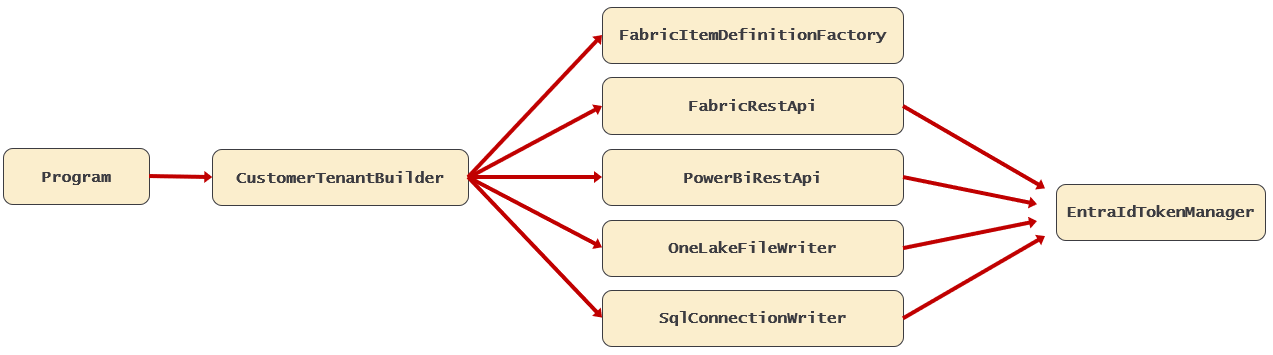
The **FabricIsvPlaybook** project contains a top-level folder name **Services** with the source files for a set of C# classes, each of which has been designed to encapsulate logic to accomplish a specific set of tasks.

A screenshot of a computer

Description automatically generated

The code in the **FabricIsvPlaybook** project has been written as a developer sample to demonstrate deploying Fabric solutions using the Fabric REST APIs. By examining and testing out the code in this project, you will build your understanding of how to deploy solutions by creating workspaces and workspace items.

The follow diagram shows how the classes in the **FabricIsvPlaybook** project interact with one another.



Here is a quick summary of each class and its purpose in the developer sample.

* **EntraIdTokenManager** contains code to authenticate with Entra Id and acquire access tokens.
* **FabricRestAPI** contains code that executes API calls using the Fabric REST API.
* **FabricItemDefinitionFactory** contains code which creates item definitions used to call Create Item.
* **PowerBiRestAPi** contains code that executes API calls using the Power BI REST API.
* **OneLakeFileWriter** contains code using ADLS GEN2 APIs to upload files to OneLake storage in lakehouse.
* **SqlConnectionWriter** contains code to execute SQL statements against SQL endpoint of warehouse.
* **CustomerTenantBuilder** contains top-level logic for deploying several examples of Fabric solutions.
* **Program** contains entry point to application. You command and uncomment code to run different tests.
* **AppLogger** contains code to write formatted output to the console window so you can see what’s going on.

The **FabricIsvPlaybook** project is a developer sample which has been created as a learning resource. The design of this project is based on several assumptions and simplifciations which make things easier to understand. Some of the code in this project needs to be hardened and/optimized before it’s used in a real-world application.

## Learning Path for Fabric Developers

Now that you’ve reached the end of the first chapter, you’ve started your journey to become a productive Fabric developer. Over the next few chapters, you’ll continue on this journey by examining sample code and learning to programming with Fabric REST APIs. However, you can’t really do anything with the Fabric REST APIs until you learn how to authenticate with the Entra Id Service to acquire access tokens. That’s why the second chapter dives into security.

The **Authentication and Authorization** chapter examines security and discusses your options for executing Fabric REST API calls as a service principal versus as a user. The chapter also discusses how using managed identities in Microsoft Azure can provide a more secure approach for executing API calls as a service principal.

The **Fabric REST API Fundamentals** chapter introduces essential concepts and programming patterns used with the Fabric REST APIs. The chapter also introduces the Fabric REST API .NET SDK which boosts developer productivity by hiding tedious and low-level details associated with directly calling Fabric REST API endpoints.

The **Program Capacities and Workspaces** chapter covers using the Fabric REST APIs to create workspaces and to associate workspaces with premium capacities. The chapter also examines how to configure access to workspace content by adding workspace role assignments for users, groups and service principals.

The **Programming Connections** chapter covers how to create connections with the Fabric REST APIs. This is an important topic because you will be required to create connections in order to connect workspace items such as shortcuts, data pipelines and semantic models to external datasources. The chapter will examine creating several different types of connections include Shareable Cloud Connections (SCCs), On-Premises Gateway connections and Virtual Gateway connections.

The **Create and Update Workspace Items** chapter covers creating workspace items using the **Create Item** API. Along the way, you’ll learn about item definitions and how they are used to create and configure workspace items. You will also learn how to update workspace items using the **Update Item Definition** as well as how to reverse engineer the item definition for a workspace item by calling the **Get Item Definition** API.

These first six chapters are the foundational chapters of this guidance document because they introduce a sequence of concepts and topics that continue to build on one another. After you have completed the first six chapters, you should feel free to skip ahead to any of the workload-specific deployment chapters that are most interesting to you.