# Program Connections

Programming connections is an essential aspect of deploying Fabric solutions. For example, you’ll be required to automate the creation and management of connections when deploying Fabric solutions that include workspace items such as OneLake shortcuts, data pipelines and semantic models.

Connections are similar to workspaces in the sense that they are both platform items as opposed to workspace items. They are scoped to the current Entra Id tenant. You configure access to connections by adding connection role assignments.

## Connection Fundamentals

As a Fabric developer, it’s important for you to distinguish between inbound security versus outbound security. **Inbound security** is involved when a custom application running outside the Fabric environment executes API calls on Fabric REST API endpoints. As you learned in earlier chapters, a custom application must first authenticate with the Entra Id Service and acquire access tokens. The custom application must then transmit an access token in each and every API call executed on a Fabric REST API endpoint.

**Outbound security** is different because it involves a scenario where you’re creating a workspace item that requires a connection to an external datasource. A connection for a workspace item must be initialized from inside the Fabric environment. For example, you can create a OneLake shortcut which connects to an ADLS Gen2 storage container. Likewise, you can create a semantic model which connects to an Azure SQL database.

A close-up of a screen

Description automatically generated

There is an important aspect of Fabric connection to keep in mind. When you create a connection, you’re not really establishing a connection across the network at that point in time. Instead, you are really just creating a persistent Fabric object with metadata that can be used to create physical connections at some point in the future. The metadata that Fabric persists for a connection includes the datasource type, the location and whatever security credentials are required to authenticate and gain access to the external datasource.

So when is the persisted metadata actually used to establish a connection across the network? A connection is established when a user or service principal accesses a workspace item such as a OneLake shortcut. Consider a scenario in which a user opens a lakehouse in the browser-based Fabric user experience. When the user clicks on the shortcut to examine the files in the ADLS Gen2 storage container, that will trigger the Fabric environment to read the persisted connection metadata to establish a physical connection to the datasource across the network.

### Connectors

Every Fabric connection is based on a specific connector. A connector is a component containing the logic to create a connection to a specific type of datasource. For example, you can use the **AzureDataLakeStorage** connector to create connections used to access content in an ADLS Gen2 storage container. You can use the **SQL** connector to create connections used to access content in an Azure SQL Database.

A close-up of several red boxes

Description automatically generated

Fabric currently supports more than 200 different connectors. The list of supported connectors and their capability are constantly evolving. Consequently, the Fabric platform provides the **List Supported Connection Types** API. This API makes it possible for a developer to discover which connectors are currently available on the platform.

For example, you can call the **List Supported Connection Types** API to retrieve metadata for a specific connector. This metadata tells you what parameters you need to pass when creating a connection based on that connector type. The metadata also tells you which types of credentials are supported by that connector.

### Metadata for Creating Connection

You can call the **Create Connection** API when you need to automate the creation of a connection. To call the **Create Connection** API, you must execute a POST request which includes JSON in the request body with the metadata required to create a connection. The JSON must include several top-level properties include **displayName**, **connectivityType**, **privacyLevel**, **connectionDetails** and **credentialDetails** as shown in the following listing.

{

"displayName": "My First Connection",

"connectivityType": "ShareableCloud",

"privacyLevel": "None",

"connectionDetails": { ... },

"credentialDetails": { ... }

}

When creating a connection, you have the option of including a **displayName**. You can alternatively leave the display name as an empty string. If you include a **displayName**, however, the value must be unique and distinct from the **displayName** of all the other existing connections on a tenant-wide basis. You will experience an error if you call the **Create Connections** API in an attempt to create a connection using a **displayName** that is already in use.

### Connection Connectivity Types

Fabric connections are split out into separated categories based on their connectivity type. **Connectivity type** is a connection property that indicates architectural details of whether a connection relies on a gateway as well as which type of gateway. As you can see in the following table, there are two cloud-based connectivity types which do not require a gateway. There are three connectivity types that indicate the requirement for a specific type of gateway.

|  |  |
| --- | --- |
| Connectivity Type Key | Connectivity Type |
| ShareableCloud | Shareable cloud connection |
| PersonalCloud | Personal cloud connection |
| VirtualNetworkGateway | Virtual gateway connection |
| OnPremisesGateway | On-prem gateway connection |
| OnPremisesGatewayPersonal | Personal on-prem gateway connection |

When creating a connection to a datasource that does not require a gateway, Microsoft recommends using **shareable cloud connections**. As its name implies, you can share a shareable cloud connectionwith other users and service principals using connection role assignments. The creator of a shareable cloud connectionis automatically assigned the **Owner** role. After creating a shareable cloud connection, you can configure other users and service principals with access by adding connection role assignment based on the roles of **User**, **User with Reshare** and **Owner**.

**Personal cloud connections** represent the original type of connection used in Power BI over the last decade. There is a problem with personal cloud connections in that they cannot be shared. Each personal cloud connection is exclusively owned and managed by a single user or service principal. You can think of personal cloud connections as a legacy type of connection that should only be used in scenarios which require backwards compatibility with semantic models.

The goal of this chapter is to teach you how to create shareable cloud connections using the Fabric REST APIs. You will learn what’s required to create shareable cloud connections to the popular datasources such as Azure Storage containers, SQL databases and data files accessible through an HTTP-based web server. Note that the scope of this chapter doesn’t include coverage of on-premises gateways or virtual gateways.

### Privacy Levels

**Privacy levels** were originally created years ago for Power BI as a security mechanism to provide datasource isolation when using the Power Query mashup engine. More specifically, Power BI datasources can be configured with a privacy level settings which indicates the degree to which the Power Query mashup engine should isolate a datasource from other datasources. You can configure a sensitive datasource with a restrictive privacy level to isolate it and to block information from being leaked to other datasources.

In Fabric, every connection is created with the **privacyLevel** property setting. Keep in mind that the **privacyLevel** property setting is only relevant in scenarios in which the Power Query mashup engine is used for data acquisition. This leads to an important observation. The **privacyLevel** property setting for connection is only relevant when the connection is used by the following workspace item types that rely on the Power Query mashup engine.

* Dataflow Gen1
* Dataflow Gen2
* Semantic models which use Import mode

If you are creating a connection to be used by one of these workspace item types, you can control the isolation level used by the Power Query mashup engine by setting the **privacyLevel** property to a value of **Public**, **Organizational** or **Private**. The following table.

|  |  |
| --- | --- |
| Privacy Level | Description |
| Public | Data retrieved can be merged together with data retrieved from any other connections. |
| Organizational | Data retrieved can be merged together with data retrieved from connections with a **privacyLevel** property setting of either **Organizational** or **Private**. Data retrieved cannot be merged together with data retrieved from connections with with a **privacyLevel** property setting of either **Public** or **None**. |
| Private | Data retrieved cannot be merged together with data retrieved from any other connections including connections with a **privacyLevel** property setting of **Private**. Used to secure data retrieved from the most sensitive datasources. |
| None | No privacy level setting is configured. Used for connections that do not use Power Query. |

Remember that the **privacyLevel** property setting for a connection is ignored in scenarios that do involve the Power Query mashup engine. For example, how should you set the **privacyLevel** property setting when creating a connection to be used by a OneLake shortcut or a data pipeline? In these scenarios, the **privacyLevel** property setting will be ignored. Therefore, it is recommended to use a **privacyLevel** property setting of **None** to avoid confusion.

### Connection Details, Creation Methods and Parameters

When preparing the JSON for creating a new connection, you must include the **connectionDetails** property with three child properties named **type**, **creationMethod** and **parameters**.

"connectionDetails": {

"type": "Web",

"creationMethod": "Web",

"parameters": [ ... ],

},

The **type** property is used to indicate which connector to use. There are over 200 supported connectors and each connector has a unique name that can be used to configure the **type** property. Examples of common connector names include **AzureDataLakeStorage**, **SQL** and **Web**.

In addition to the **type** property, the **connectionDetails** section contain two other important properties named **creationMethod** and **parameters**. Before you can understand how to set these properties, you must learn background information about the concept of a creation method.

A ***creation method*** defines the set of named parameters used to create a connection for a specific connector type. Every connector is required to support at least one creation method. For example, the **Web** connector provides a creation method named **Web** which defines a parameters list with a single text parameter named **url**.

"connectionDetails": {

"type": "Web",

"creationMethod": "Web",

"parameters": [

{

“name": "url",

"dataType": "Text",

"value": "https://github.com/FabricDevCamp/ProductSalesData/raw/main/data”

}

],

},

As you can see for the previous listing, the **creationMethod** property value determines what parameters should be included in the **parameters** section. The **parameters** section is configured with an JSON array of parameters where each parameter is configured with the **name** property, the **dataType** property and the **value** property.

Now let’s look at an example using another connector. The **SQL** connector provides a creation method named **Sql** which defines a parameters list with two text parameters named **server** and **database**. You can create a connection to an Azure SQL database by constructing the **connectionDetails** section as shown in the following JSON listing.

"connectionDetails": {

"type": "SQL",

"creationMethod": "Sql",

"parameters": [

{

"name": "server",

"dataType": "Text",

"value": "devcamp.database.windows.net"

},

{

"name": "database",

"dataType": "Text",

"value": "WingtipSales"

}

]

}

Let’s examine one more example using a third connector. The **AzureDataLakeStorage** connector provides a creation method named **AzureDataLakeStorage** which defines a parameters list with two text parameters named **server** and **path**.

"connectionDetails": {

"type": "AzureDataLakeStorage",

"creationMethod": "AzureDataLakeStorage",

"parameters": [

{

"name": "server",

"dataType": "Text",

"value": "https://devcampdemos.dfs.core.windows.net",

},

{

"name": "path",

"dataType": "Text"

"value": "/",

}

]

},

The high-level design used by connectors to configure creation methods make it possible for a connector to define two or more creation methods. However, you will find all the connectors currently supported in Fabric only support a single creation method. In most cases, the connector authors have defined a single creation method using the same name as the connector itself. However, there are some connectors with creation methods whose name differs from the connector name.

### Credential Details and Credential Types

When calling the **Create Connection** API, the creation request must include metadata to indicate which type of credentials should be used when connecting to the target datasource. The metadata for configuring connection credentials must be included in the **credentialDetails** section as shown in the following JSON listing.

"credentialDetails": {

"credentials": {

"credentialType": "Anonymous"

},

"connectionEncryption": "NotEncrypted",

"skipTestConnection": false,

"singleSignOnType": "None"

},

The **credentialDetails** property requires a child property named **credentials**. In turn, the **credentials** property requires a child property named **credentialType**. The **credentialType** property must be configured with one of the credential types supported in Fabric which are listed in the following table.

|  |  |
| --- | --- |
| Credential Type | Description |
| Anonymous |  |
| Basic |  |
| OAuth2 |  |
| Key |  |
| SharedAccessSignature |  |
| ServicePrincipal |  |
| Windows |  |
| WindowsWithoutImpersonation |  |
| WorkspaceIdentity |  |

In the previous example in which the **credentialType** property is set to **Anonymous**, there is no need to provide any additional properties in the **credentials** section because there is no need to supply any actual credentials.

"credentials": {

"credentialType": "Anonymous"

}

However, you will need to include additional properties inside the **credentials** section when using credential types other than **Anonymous**. Consider a simple example of creating a connection using an Azure storage account key as its credentials. When setting the **credentialType** property to **Key**, you must also include the **key** property so you can configure the key credentials for the connection.

"credentials": {

"credentialType": "Key"

"key": "{AZURE\_STORAGE\_ACCOUNT\_KEY\_VALUE}"

}

When setting the **credentialType** property to **Basic**, you must also include the **username** and **password** properties so you can configure credentials to a datasource that supports basic authentication such as an Azure SQL database.

"credentials": {

"credentialType": "Basic",

"username": "SqlUser01",

"password": "{SQL\_USER\_PASSWORD}",

}

In addition to the **credentials** property, the **credentialDetails** section requires three other properties named **connectionEncryption**, **skipTestConnection** and **singleSignOnType**. It’s important that you understand when and how these properties affect a connection’s behavior.

The **connectionEncryption** property indicates whether data moving through the connection should be encrypted by the server. This type of connection encryption is supported by SQL-based datasources such as SQL Server, Azure SQL, My SQL and PostgreSQL. Connectors for these datasource support **connectionEncryption** property values of **NotEncrypted** or **Encrypted**. Many connectors do not support this type of encrypted connection and therefore only support a **connectionEncryption** property value of **NotEncrypted**.

The **skipTestConnection** is a property which is used to enable a feature to bypass the automatic connectivity test when creating a connection. A value of true is only supported handful of connectors including Web and OData connectors. Enabling this feature allows you to proceed even in a scenario where the initial connection attempt fails and it typically used when you know the datasource is temporarily unavailable or when you want to quickly create a connection and test later.

The **singleSignOnType** property indicates whether the connection is configured to support single sign on. The main idea of single sign on is that the current user’s identity is propagated through the connection and is used to connect to datasource. In a concurrent environment with multiple users, each physical connection is created dynamically based on the identity of the current user.

When creating connections for a Fabric solution, it’s a best practice to avoid using single sign on and instead to set connection credentials based on fixed identity. As Fabric users use the connection directly or indirectly, the same fixed identity is used whenever Fabric establishes a physical connection across the network. That means that the most common **singleSignOnType** property value is **None**.

In many cases, the three properties named **connectionEncryption**, **skipTestConnection** and **singleSignOnType** will have no relevance for the connection you are creating. However, you still need to include these three properties in a request to create a connection using the property values shown in the following JSON listing.

"credentialDetails": {

"credentials": { ... },

"connectionEncryption": "NotEncrypted",

"skipTestConnection": false,

"singleSignOnType": "None"

}

### Discover Supported Connection Type Capabilities

Ddddd

{

"type": "Web",

"creationMethods": [

{

"name": "Web",

"parameters": [

{ "name": "url", "dataType": "Text", "required": true, "allowedValues": null }

]

}

],

"supportedCredentialTypes": [ "Anonymous", "Basic", "OAuth2", "ServicePrincipal" ],

"supportedConnectionEncryptionTypes": [ "NotEncrypted" ],

"supportsSkipTestConnection": true

},

xx

{

"type": "SQL",

"creationMethods": [

{

"name": "Sql",

"parameters": [

{ "name": "server", "dataType": "Text", "required": true, "allowedValues": null },

{ "name": "database", "dataType": "Text", "required": false, "allowedValues": null }

]

}

],

"supportedCredentialTypes": [ "Basic", "OAuth2", "ServicePrincipal" ],

"supportedConnectionEncryptionTypes": [ "NotEncrypted", "Encrypted" ],

"supportsSkipTestConnection": false

},

C

{

"type": "AzureDataLakeStorage",

"creationMethods": [

{

"name": "AzureDataLakeStorage",

"parameters": [

{ "name": "server", "dataType": "Text", "required": true, "allowedValues": null },

{ "name": "path", "dataType": "Text", "required": true, "allowedValues": null }

]

}

],

"supportedCredentialTypes": [ "Key", "OAuth2", "SharedAccessSignature", "ServicePrincipal", "WorkspaceIdentity" ],

"supportedConnectionEncryptionTypes": [ "NotEncrypted" ],

"supportsSkipTestConnection": false

}

Xx

"allowedValues": null

xxx

"allowedValues": [ "enabled", "disabled" ]

xxx

### Create Connections by Hand

Before you attempt to automate the creation of connections using code, it can be helpful to create a few connections by hand. This exercise will give you a sense of what decisions you need to make whenever you need to create a new connection.

Remember that connections are platform items that exist within the scope of the current Entra Id tenant. That means that Fabric connection can be shared across workspaces. Of course, just because you can doesn’t mean that you should.

You can see all the connections you have access to using the Manage Connections and Gateways page in the Fabric Service.

A screenshot of a computer

Description automatically generated

Dddddd

A screenshot of a computer

Description automatically generated

Xxxx

A screenshot of a computer

Description automatically generated

Xxxx

A screen shot of a computer

Description automatically generated

xxx

A screenshot of a computer

Description automatically generated

xxxx

## Program Connection CRUD Operations

Ll,l,,

### Create Connection

### List Connections

Ccccc

### Get Connection

### Update Connection

### Delete Connection

## Program Connection Role Assignments

### Add Connection Role Assignment

### List Connection Role Assignments

### Delete Connection Role Assignment

## Create Connections for Common Scenarios

Content to come

### Create a Azure SQL Connections using Basic Credentials

Content to come

### Create an Azure Storage Connection using Service Principal Credentials

Content to come

### Create an Azure Storage Connection using Workspace Identity

Content to come

### Create a OneLake Shortcut using a Connection

Content to come

### Bind a Connection to a Semantic Model

Content to come