Using Data Factory Parameterised Linked Services

mrpaulandrew.com/2018/11/15/using-data-factory-parameterised-linked-services

View all posts by mrpaulandrew November 15, 2018

Microsoft recently announced that we can now make our Azure Data Factory (ADF) v2 pipelines *even more* dynamic with the introduction of **parameterised Linked Services**. This now completes the set for our core Data Factory components meaning we can now inject parameters into every part of our Data Factory control flow orchestration processes.

- Pipelines
- Activities
- Datasets
- Linked Services

Currently parameter support is limited to only a handful of common Linked Services, listed on the below Microsoft documents page:

https://docs.microsoft.com/en-us/azure/data-factory/parameterize-linked-services

In this post I wanted to have a play with this new capability myself and show you how to bubble up our new Linked Service parameters in a chain of dynamic content all the way up to the Pipeline level and beyond

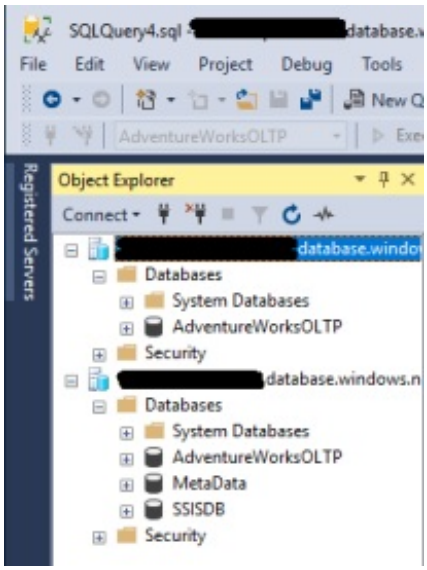
Scenario

For this 'how to' guide we are going to keep things simple, I'm using Azure SQL Database's as the source and sink for my pipeline.

- On my first logical SQL instance I have a complete sample Adventure Works database.
- On my second logical SQL instance I have an empty Adventure Works database. Also, for ease I've remove any computed columns and foreign keys on the target.

With Data Factory I'm going to create a dynamic pipeline that copies data from one set of database tables to the other… Basically setting up a SQLDB read only replica using Data Factory, don't judge me!

After the first simple pipeline is in place we can then have a little more fun driving this with a metadata driven pipeline that bootstraps the process to copy all tables from source to sink, in my case using our source SQLDB sys.objects system table.



Concept

Just before we drive in…

Above I've listed our Data Factory components in a deliberate order and I also talked about bubbling up our new Linked Service parameters to the pipeline level. The reasoning here and what we need to understand is that everything in Data Factory starts with a Linked Service connection. If this content is now going to be dynamic as well, this doesn't leave anything for us to build on/test/preview when we add our subsequent Data Factory components.

To clarify, we are making our lowest level ADF component dynamic. But we want to trigger our process at the highest level in the Pipeline. We therefore need to create a chain of variables through each ADF component level to allow this translation to happen.

In the case of my SQLDB scenario above:

- In the **Linked Service** I add a parameter to be used for the logical SQL instance attribute.
- I then create a **Dataset**, when referencing the Linked Service this prompts for the logical SQL instance parameter. I add another parameter to the Dataset that can be used for the Linked Service parameter.
- I then create a Pipeline and add a Copy **Activity**. The copy activity source references my single dataset and prompts for the logical SQL instance parameter again. So I create another **Pipeline** parameter to feed that.

To try and visualise that because we are only using a single Linked Service and single Dataset:

What this means is at runtime our pipeline level parameter gets passed back down through the component layers, eventually setting the Linked Service SQL Instance attribute as intended (the orange arrow).

Pipeline param > Activity Dataset parma > Linked Service param > SQL Instance Attribute.

Great, I'm glad we've got that cleared up. Now let's build something!

Blog Supporting Content also in my **GitHub** repository:

https://github.com/mrpaulandrew/BlobSupportingContent

Simple Pipeline

Following the above approach above I've firstly created a generic Azure SQLDB Linked Service in my Data Factory and parameterised the following attributes:

- SQL Instance
- Database Name
- SQL User
- SQL Password

Data Factory Linked Service wizard screen shot below.

Next, in my Dataset I've repeated the above parameters and also added a fifth parameter for the database 'Table Name'. Seen below.

Lastly, in my Pipeline Copy Activity I then need another set, plus, this time I need to double up on those 5 parameters. Creating a set for the Source and a set for the Sink of the copy. Seen below.

This means that at runtime I have a single pipeline, single activity, single dataset and single linked service that prompts for 10 parameter values to perform a copy of some SQLDB table. All these parameters are strings as you can see in the screen shot below when I triggered the pipeline in debug mode (click to enlarge).

This works allowing me to manually input all 10 values at runtime to perform the copy of one SQLDB table from one instance to another. Or, even the same instance. It's all dynamic so whatever!

Metadata Driven Pipeline

Now let's extend our simple pipeline created above to perform the copy for all tables in a given source SQL instance using some metadata. This isn't a new pattern, but I thought it worth doing again with a dynamic Linked Service, simply to prove that a single Linked Service and single Dataset can be used for a parallel copy of multiple tables with a different Source and Sink SQL instances.

This pipeline more complex pipeline contains:

- A Lookup activity to get a list of tables from our source SQL instance using the same dataset and linked service.
- Then a ForEach activity to parallelise the copy operation. Within the ForEach:
    - I have a Stored Procedure activity to Truncate my target tables. Again, same Linked Service and Dataset used.
    - A Copy activity doing the same thing as my simple pipeline above. But now with the table name provided from the Lookup activity array of items.

For the Lookup activity I passed a simple T-SQL query to sys.objects with a WHERE clause for the Adventure Works 'SalesLT' schema.

For the Stored Procedure activity I used sys.sp_executesql to TRUNCATE the target tables passing the table name from the Lookup activity array of items. Remember I remove all foreign keys in the target Adventure Works database. See setting screen shot below.

This is simply doing the equivalent of the following T-SQL, just wrapped up in an ADF activity with the dynamic values injected using JSON variables:

```
DECLARE @SQL NVARCHAR(MAX) = '@concat('TRUNCATE TABLE ',item().TableName)'

EXEC sys.sp_executesql @statement = @SQL
```

Finally, the pipeline can be triggered using the same approach as the simple pipeline with parameters requested at runtime for the source and target SQL instance.... we've got to hardcode these somewhere! Of course, we could call out to a third SQLDB if you want to add another layer of abstraction to fetch the connection details. Or, maybe a Web activity that hits an Azure Function, which hits Azure Key Vault to return the connection info. Where does the abstraction end

All the above code for these example pipelines is in my GitHub repo.

Blog Supporting Content also in my **GitHub** repository:

https://github.com/mrpaulandrew/BlobSupportingContent

I hope you found this post useful.

Many thanks for reading.