Data Savvy

- My experiences and education in data modeling, integration, transformation, analysis, and visualization -

Category: Azure Data Factory

AZURE, **AZURE DATA FACTORY**, **MICROSOFT TECHNOLOGIES**, **POWER BI**

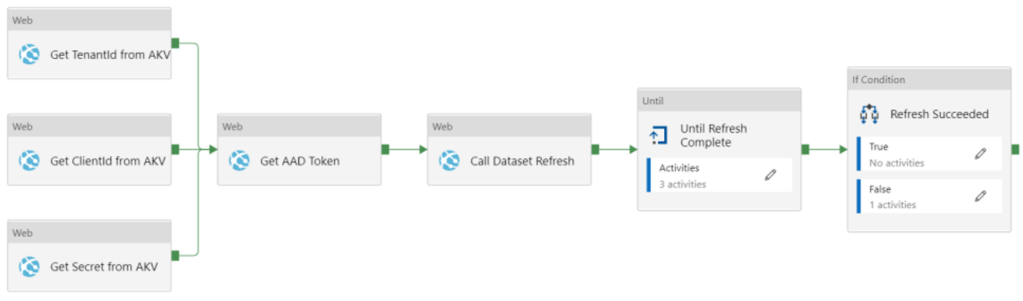
Refreshing a Power BI Dataset in Azure Data Factory

July 9, 2020July 10, 2020 Meagan Longoria

I recently needed to ensure that a Power BI imported dataset would be refreshed after populating data in my data mart. I was already using Azure Data Factory to populate the data mart, so the most efficient thing to do was to call a pipeline at the end of my data load process to refresh the Power BI dataset.

Power BI offers REST APIs (https://docs.microsoft.com/en-us/rest/api/power-bi/) to programmatically refresh your data. For Data Factory to use them, you need to register an app (service principal) in AAD and give it the appropriate permissions in Power BI and to an Azure key vault.

I'm not the first to tackle this subject. Dave Ruijter has a great blog post (https://www.moderndata.ai/2019/05/powerbi-dataset-refresh-using-adf/) with code and a step-by-step explanation of how to use Data Factory to refresh a Power BI dataset. I started with his code and added onto it. Before I jump into explaining my additions, let's walk through the initial activities in the pipeline.



Refresh Power BI Dataset Pipeline in Data Factory

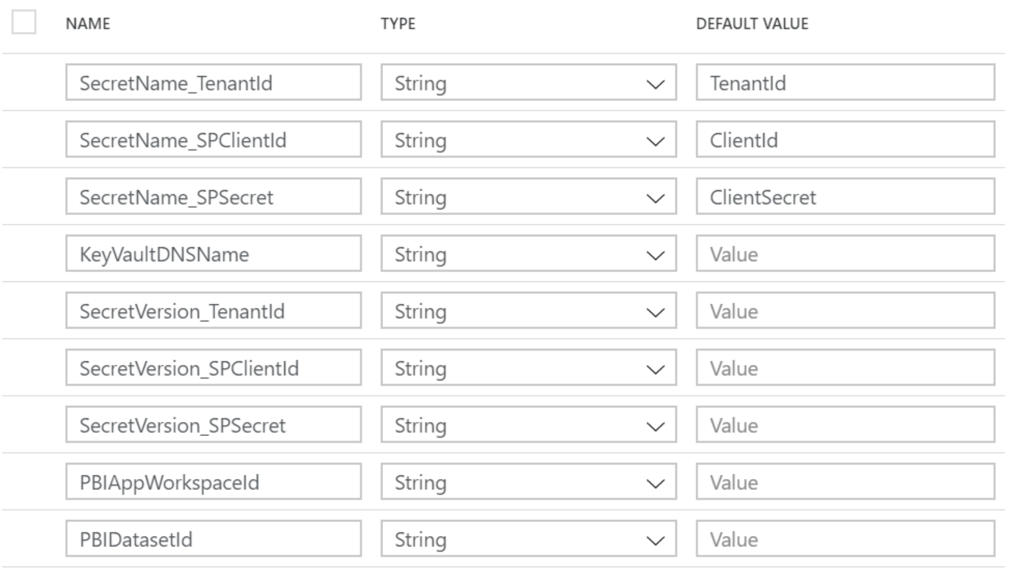
Before you can use this pipeline, you must have:

- an app registration in Azure AD with a secret
- a key vault that contains the Tenant ID, Client ID of your app registration, and the secret from your app registration as separate secrets.
- granted the data factory managed identity access to the keys in the key vault
- allowed service principals to use the Power BI REST APIs in in the Power BI tenant settings

- granted the service principal admin access to the workspace containing your dataset

For more information on these setup steps, read Dave's post.

The pipeline contains several parameters that need to be populated for execution.



NAME	TYPE	DEFAULT VALUE
SecretName_TenantId	String	TenantId
SecretName_SPClientId	String	ClientId
SecretName_SPSecret	String	ClientSecret
KeyVaultDNSName	String	Value
SecretVersion_TenantId	String	Value
SecretVersion_SPClientId	String	Value
SecretVersion_SPSecret	String	Value
PBIAppWorkspaceId	String	Value
PBIDatasetId	String	Value

ADF pipeline parameters

The first seven parameters are related to the key vault. The last two are related to Power BI. You need to provide the name and version of each of the three secrets in the key vault. The KeyVaultDNSName should be https://mykeyvaultname.vault.azure.net/ (https://mykeyvaultname.vault.azure.net/) (replace mykeyvaultname with the actual name of your key vault). You can get your Power BI workspace ID and dataset ID from the url when you navigate to your dataset settings.

The "Get TenantId from AKV" activity retrieves the tenant ID from the key vault. The "Get ClientId from AKV" retrieves the Client ID from the key vault. The "Get Secret from AKV" activity retrieves the app registration secret from the key vault. Once all three of these activities have completed, Data Factory executes the "Get AAD Token" activity, which retrieves an auth token so we can make a call to the Power BI API.

One thing to note is that this pipeline relies on a specified version of each key vault secret. If you always want to use the current version, you can delete the SecretVersion_TenantID, SecretVersion_SPClientID, and SecretVersion_SPSecret parameters. Then change the expression used in the URL property in each of the three web activities .

For example, the URL to get the tenant ID is currently:

```
@concat(pipeline().parameters.KeyVaultDNSName,'secrets/',pipeline().parameters.Se
cretName_TenantId,'/',pipeline().parameters.SecretVersion_TenantId,'?api-
version=7.0')
```

To always refer to the current version, remove the slash and the reference to the SecretVersion_TenantID parameter so it looks like this:

```
@concat(pipeline().parameters.KeyVaultDNSName,'secrets/',pipeline().parameters.Se
cretName_TenantId,'?api-version=7.0')
```

The “Call Dataset Refresh” activity is where we make the call to the Power BI API. It is doing a `POST` to `https://api.powerbi.com/v1.0/myorg/groups/ (https://api.powerbi.com/v1.0/myorg/groups/) {groupId}/datasets/{datasetId}/refreshes` and passes the previously obtained auth token in the header.

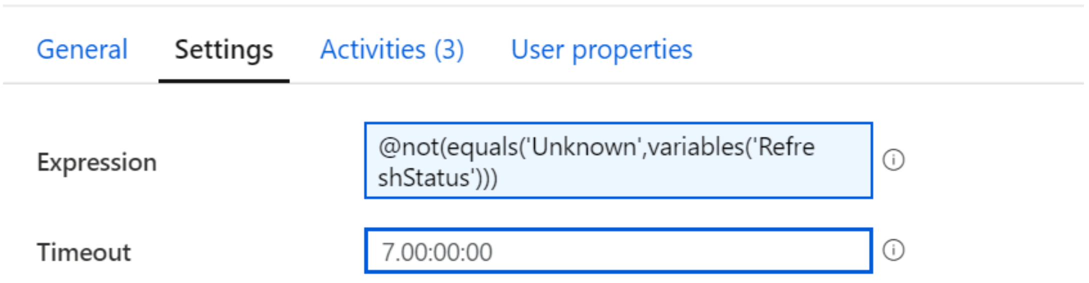
This is where the original pipeline ends and my additions begin.

Getting the Refresh Status

When you call the Power BI API to execute the data refresh, it is an asynchronous call. This means that the ADF activity will show success if the call is made successfully rather than waiting for the refresh to complete successfully.

We have to add a polling pattern to periodically check on the status of the refresh until it is complete.

We start with an until activity. In the settings of the until loop, we set the expression so that the loop executes until the RefreshStatus variable is not equal to “Unknown”. (I added the RefreshStatus variable in my version of the pipeline with a default value of “Unknown”.) When a dataset is refreshing, “Unknown” is the status returned until it completes or fails.

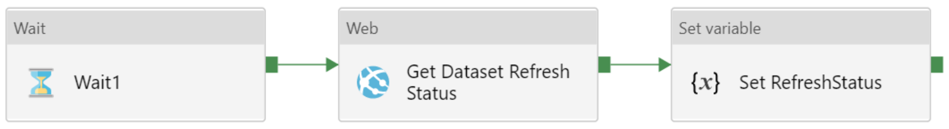


Expression: @not(equals('Unknown',variables('RefreshStatus')))

Timeout: 7.00:00:00

ADF Until activity settings

Inside of the “Until Refresh Complete” activity are three inner activities.



ADF Until activity contents

The “Wait1” activity gives the dataset refresh a chance to execute before we check the status. I have it configured to 30 seconds, but you can change that to suit your needs. Next we get the status of the refresh.

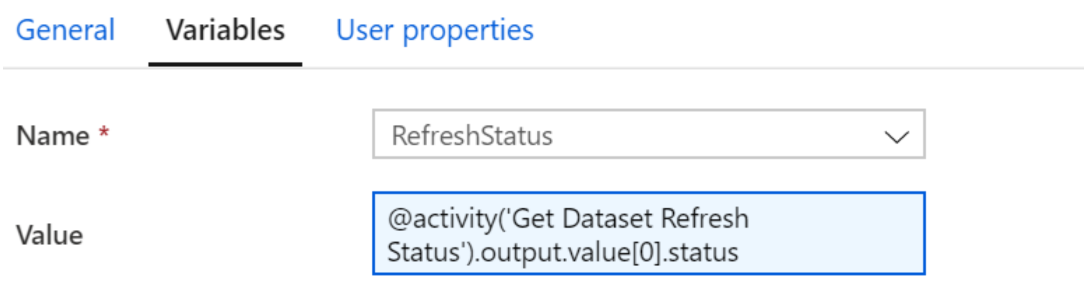
This web activity does a `GET` to the same url we used to start the dataset refresh, but it adds a parameter on the end.

```
https://docs.microsoft.com/en-us/resGET (https://docs.microsoft.com/en-us/resGET)
https://api.powerbi.com/v1.0/myorg/groups/
(https://api.powerbi.com/v1.0/myorg/groups/)
{groupId}/datasets/{datasetId}/refreshes?$top={$top}
```

The API doesn't accept a request ID for the newly initiated refresh, so we get the last initiated refresh by setting top equal to 1 and assume that is the refresh for which we want the status.

The API provides a JSON response containing an array called `value` with a property called `status`.

In the "Set RefreshStatus" activity, we retrieve the status value from the previous activity and set the value of the RefreshStatus variable to that value.



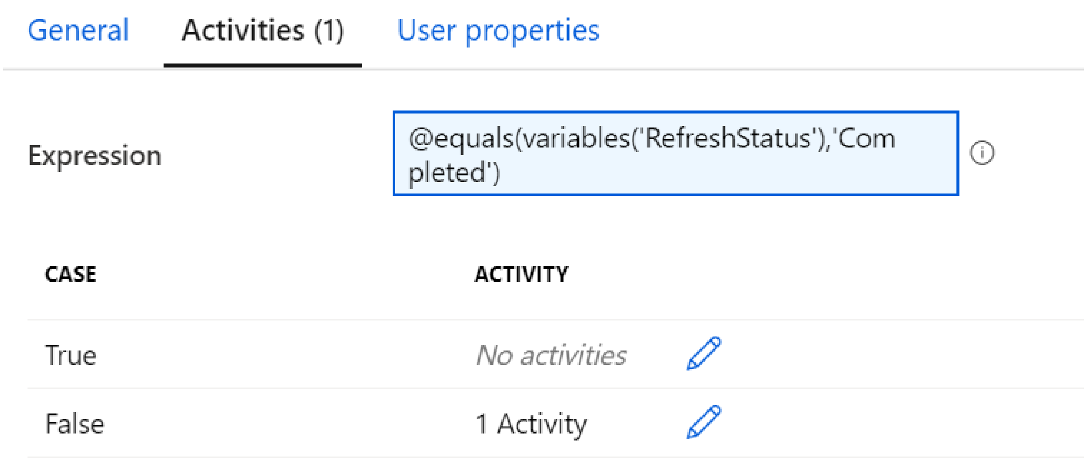
Name *	RefreshStatus
Value	@activity('Get Dataset Refresh Status').output.value[0].status

Setting the value of the RefreshStatus variable in the ADF pipeline

We want the status value in the first object in the value array.

The until activity then checks the value of the RefreshStatus variable. If your dataset refresh is complete, it will have a status of "Completed". If it failed, the status returned will be "Failed".

The If activity checks the refresh status.



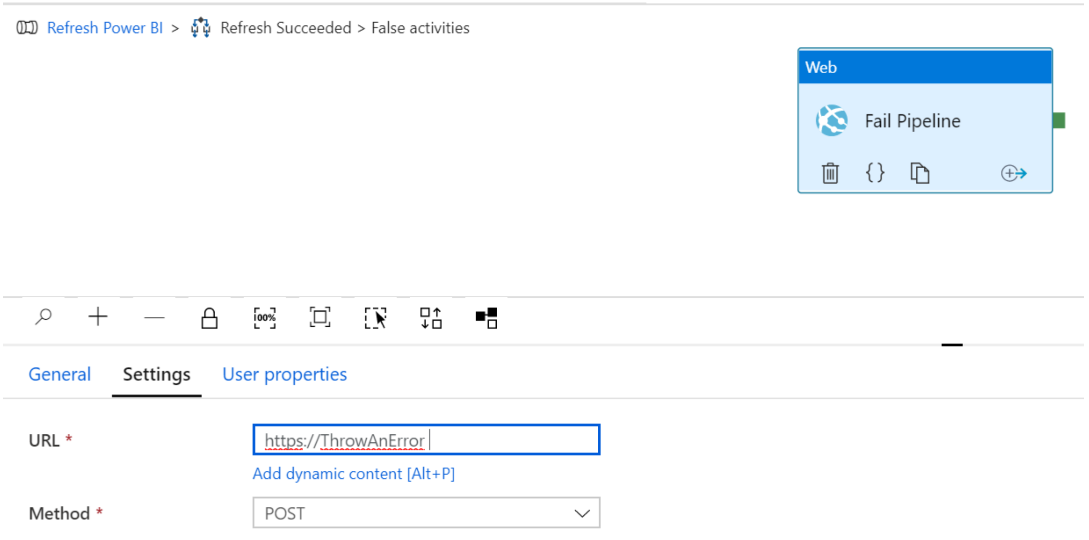
Expression: @equals(variables('RefreshStatus'),'Completed')

CASE	ACTIVITY
True	*No activities*
False	1 Activity

If activity expression in the ADF pipeline

If the refresh status is "Completed", the pipeline execution is finished. If the pipeline activity isn't "Completed", then we can assume the refresh has failed. If the dataset refresh fails, we want the pipeline to fail.

There isn't a built-in way to cause the pipeline to fail so we use a web activity to throw a bad request.



URL * https://ThrowAnError
Add dynamic content [Alt+P]
Method * POST

We do a `POST` to an invalid URL. This causes the activity to fail, which then causes the pipeline to fail.

Since this pipeline has no dependencies on datasets or linked services, you can just grab my code from GitHub and use it in your data factory (https://github.com/DC-AC/DataFactory/blob/master/Refresh%20PBI%20Dataset.json).

1 Comment
AZURE, **AZURE DATA FACTORY**, **LOGIC APPS**, **MICROSOFT TECHNOLOGIES**

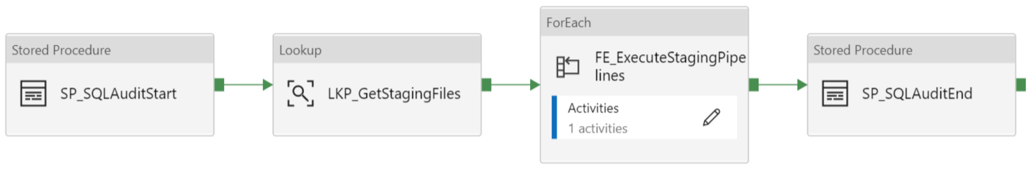
Using Logic Apps in a Data Factory Execution Framework – Part 1

May 7, 2020May 7, 2020 Meagan Longoria
Data Factory allows parameterization in many parts of our solutions. We can parameterize things such as connection information in linked services as well as blob storage containers and files in datasets. We can also parameterize certain properties in activities. For instance, we can write an expression to determine the stored procedure to be executed in a Stored Procedure Activity or the filename in the sink (destination) of a Copy Activity.

But we cannot parameterize the invoked pipeline in an Execute Pipeline Activity. This means we need to find workarounds in order to have a metadata-driven execution framework. What I mean by metadata-driven execution framework is that data is stored in a datastore (in my case, a SQL Database) and used to determine what pipelines and activities get executed. With this type of framework, if I don't want a specific pipeline to execute, I would just update my data in the datastore rather than delete the pipeline execution from the parent pipeline. We've been doing this type of development in SSIS for years, and Biml (http://bimlscript.com/) has played a big part in that. But SSIS allows us to parameterize the Execute Package Task.

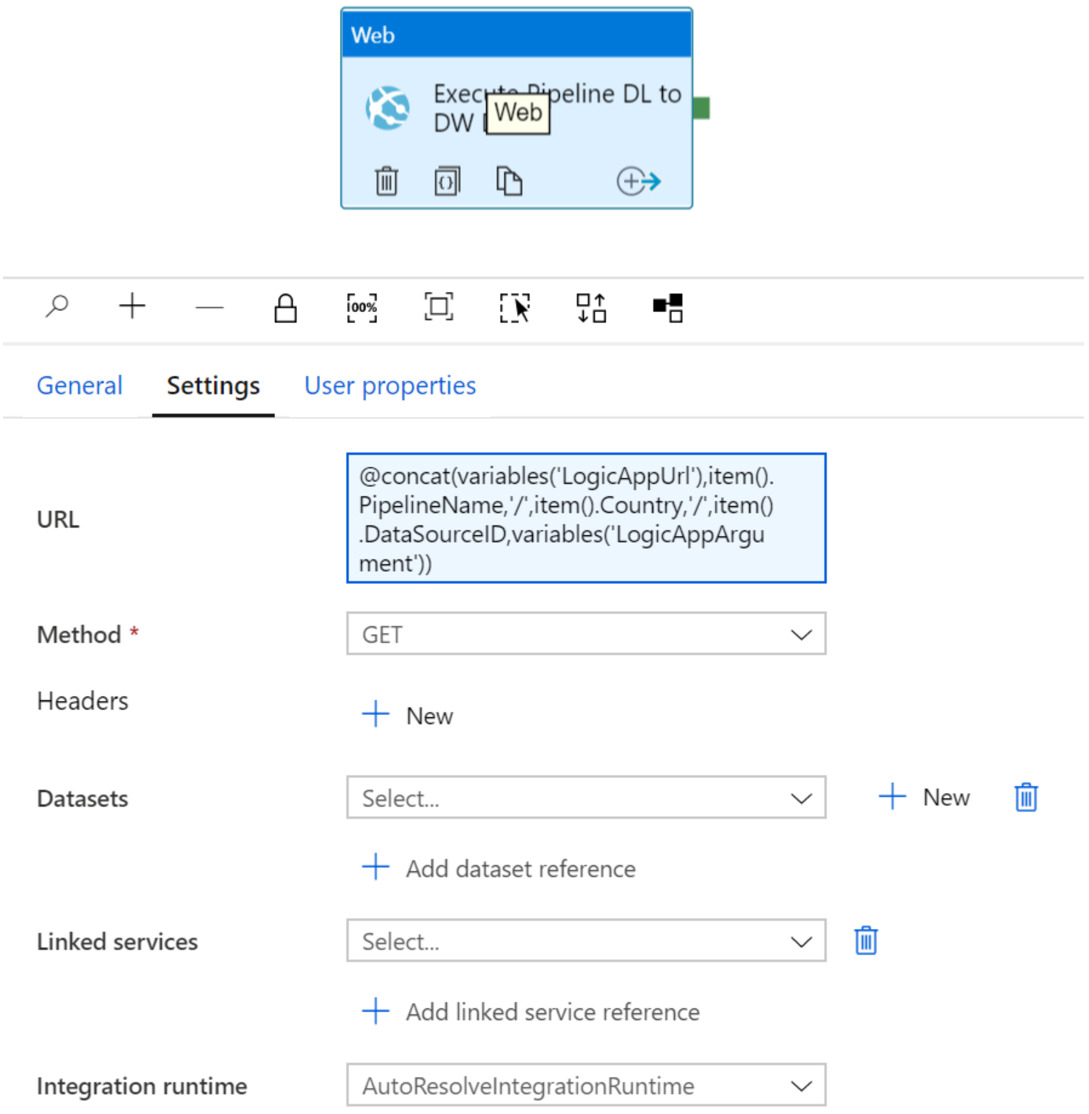
Since we can't implement this parameterized execution of pipelines natively, we need to look for something that Data Factory can call to accomplish the task. Paul Andrew has a nice framework that uses Azure Functions (https://mrpaulandrew.com/2020/02/25/creating-a-simple-staged-metadata-driven-processing-framework-for-azure-data-factory-pipelines-part-1-of-4/). I was working on a Data Factory solution for a client who doesn't have C# or PowerShell developers on hand to help with the ELT process, so we needed to explore a low-code solution.

While there is no Logic App activity in Data Factory, we can use a Web Activity to call the Logic App. I might have a pipeline that looks something like what is pictured below.



Staging pipeline that copies files from Azure Data Lake Storage to Azure SQL Database

Within the ForEach loop is a single Web Activity.



URL: @concat(variables('LogicAppUrl'),item().PipelineName,'/',item().Country,'/',item().DataSourceID,variables('LogicAppArgument'))

Method *: GET

Headers: + New

Datasets: Select... + New

+ Add dataset reference

Linked services: Select...

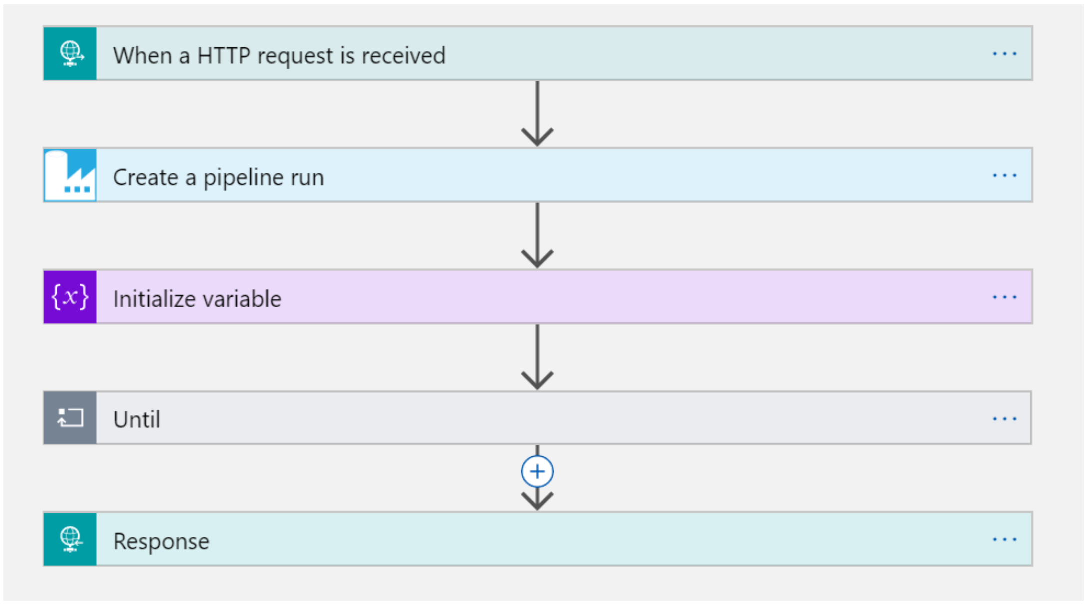
+ Add linked service reference

Integration runtime: AutoResolveIntegrationRuntime

Web Activity that calls a Logic App

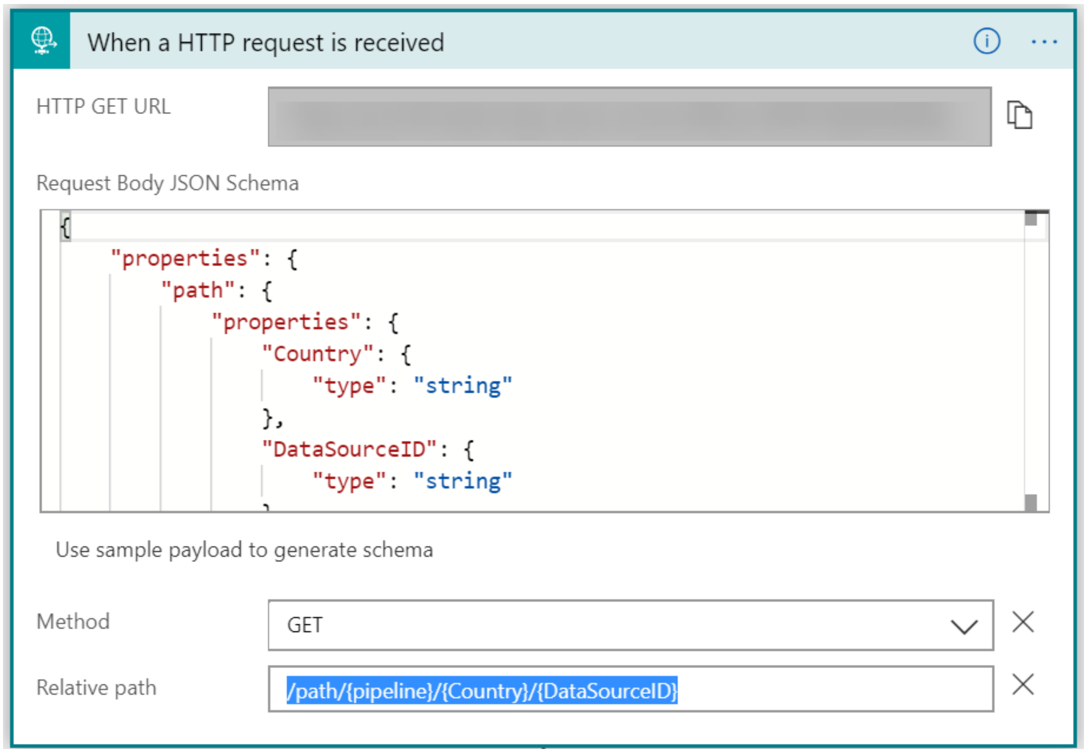
I used some variables and parameters in an expression to populate the URL so it would be dynamic. I used a GET method in the call.

My initial version of my Logic App is shown below.



Logic App that executes a Data Factory pipeline and waits for it to complete before returning a response

I added path parameters in my HTTP request trigger to allow me to capture the information I need to execute the appropriate pipeline. For me this included the pipeline name, a data source ID, and a country. Your parameters would vary according to your requirements.


When a HTTP request is received

HTTP GET URL

Request Body JSON Schema

```
{
    "properties": {
        "path": {
            "properties": {
                "Country": {
                    "type": "string"
                },
                "DataSourceID": {
                    "type": "string"
```

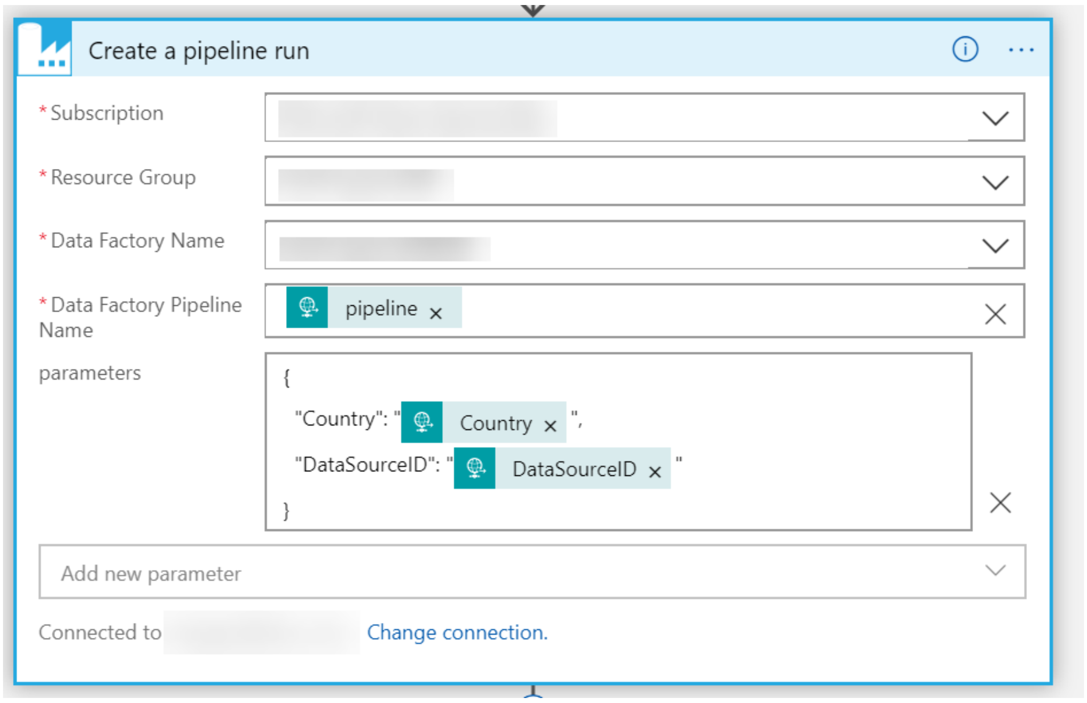
Use sample payload to generate schema

Method: GET

Relative path: /path/{pipeline}/{Country}/{DataSourceID}

HTTP Request trigger in my Logic App

Logic apps has an action called "Create a pipeline run". You tell it which data factory, which pipeline, and any parameter values needed for the pipeline execution.


Create a pipeline run

* Subscription

* Resource Group

* Data Factory Name

* Data Factory Pipeline Name: pipeline

parameters:
```
{
  "Country": "Country",
  "DataSourceID": "DataSourceID"
}
```

Add new parameter

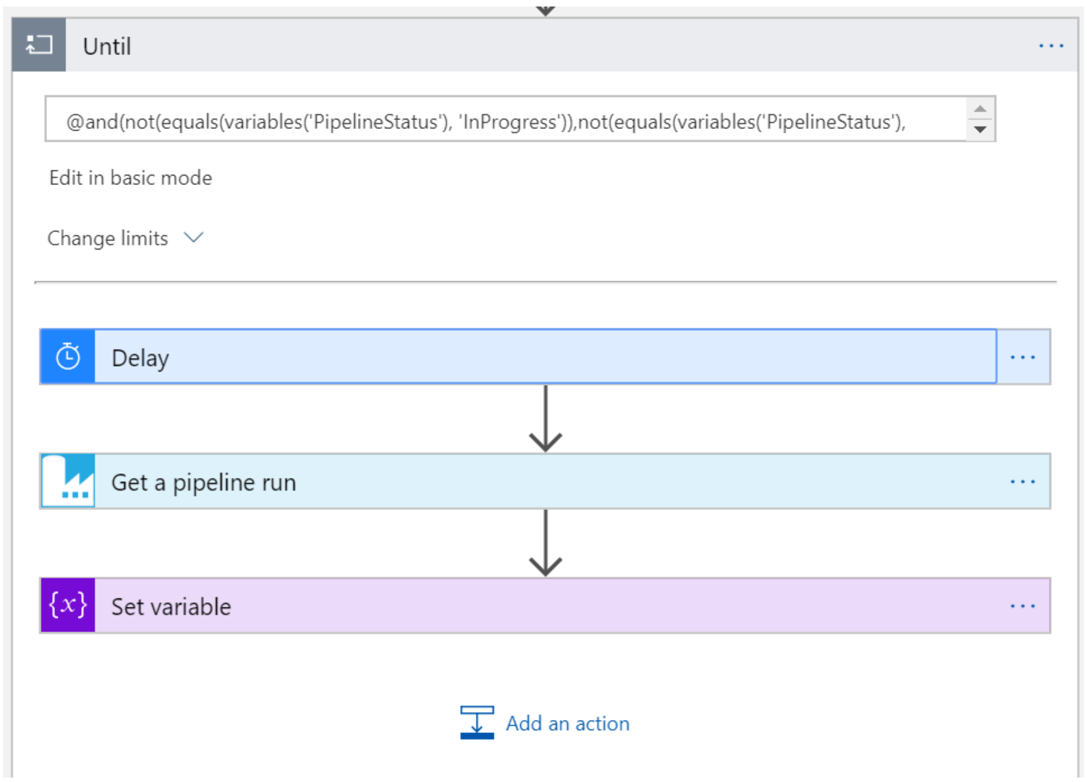
Connected to Change connection.

Create a pipeline run action in my Logic App

At this point in the workflow, our pipeline would be executing. But now we need to know when it has finished. That's what the Initialize Variable and Until Loop actions are handling. I created a string variable called Pipeline Status and set the default value to "InProgress". My Until loop action checks my pipeline execution status. If it's still running, it waits 5 seconds, gets the new status, and assigns that status to the variable. This repeats until the pipeline execution is no longer in progress.

Here's the expression I used to check whether the pipeline execution is still running:

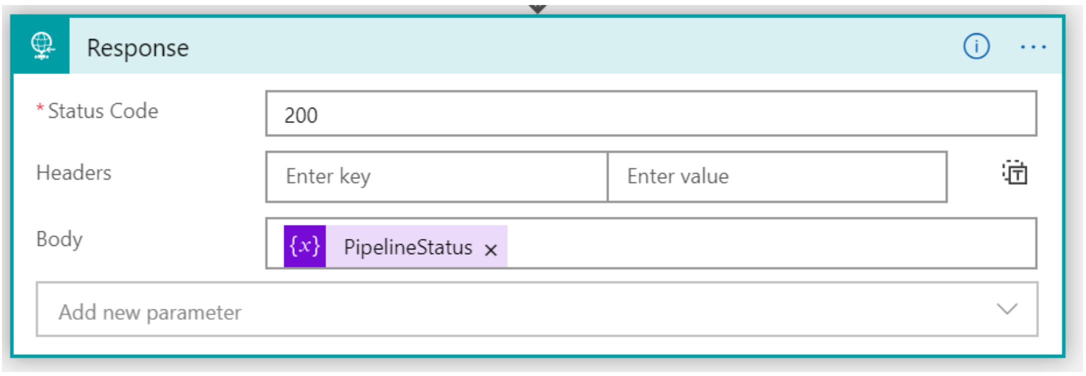
```
@and(not(equals(variables('PipelineStatus'), 'InProgress')),
not(equals(variables('PipelineStatus'), 'Queued')))
```


Until

```
@and(not(equals(variables('PipelineStatus'), 'InProgress')),not(equals(variables('PipelineStatus'),
```

Until loop in my Logic App to dynamically execute a Data Factory pipeline

Once the pipeline execution is complete, an HTTP response with the pipeline status is sent back to the caller.


Response

* Status Code	200	
Headers	Enter key	Enter value
Body	{x} PipelineStatus ×	

HTTP Response action in my Logic App

This is all great until you find out that **Logic Apps will experience an HTTP timeout if the request takes more than 2 minutes (https://www.serverless360.com/blog/use-webhooks-for-async-http-messaging-in-azure-logic-apps)**.

Do you have any pipelines that take longer than two minutes to execute? If so, you need to change your solution to handle this. Note that you would have the same issue with Azure Functions (https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale#timeout), although it would give you 230 seconds instead of 120 seconds before it timed out. We need to switch to an asynchronous call to support long running pipelines. Paul has already done this in his framework (https://mrpaulandrew.com/2020/04/20/adf-procfwk-v1-4-enhancements-for-long-running-pipelines/) using Azure Functions. In Logic Apps, we can change our response to an asynchronous response and then implement a polling pattern to check the status. We could alternatively implement a webhook action (https://www.serverless360.com/blog/use-webhooks-for-async-http-messaging-in-azure-logic-apps). I'll write about updating the solution to handle long running pipelines in a future post.

2 Comments
AZURE, **AZURE DATA FACTORY**, **MICROSOFT TECHNOLOGIES**

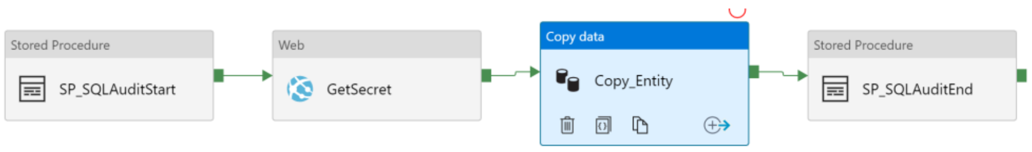
Parameterizing a REST API Linked Service in Data Factory

January 30, 2020January 30, 2020 Meagan Longoria
We can now pass dynamic values to linked services at run time in Data Factory. This enables us to do things like connecting to different databases on the same server using one linked service. Some linked services in Azure Data Factory can be parameterized through the UI (https://docs.microsoft.com/en-us/azure/data-factory/parameterize-linked-services#supported-data-stores). Others require that you modify the JSON to achieve your goal.

Recently, I needed to parameterize a Data Factory linked service pointing to a REST API. At this time, REST APIs require you to modify the JSON yourself.

In order to pass dynamic values to a linked service, we need to parameterize the linked service, the dataset, and the activity.

I have a pipeline where I log the pipeline start to a database with a stored procedure, lookup a username in Key Vault, copy data from a REST API to data lake storage, and log the end of the pipeline with a stored procedure. My username and password are stored in separate secrets in Key Vault, so I had to do a lookup with a web activity to get the username. The password is retrieved using Key Vault inside the linked service. Data Factory doesn't currently support retrieving the username from Key Vault so I had to roll my own Key Vault lookup there.

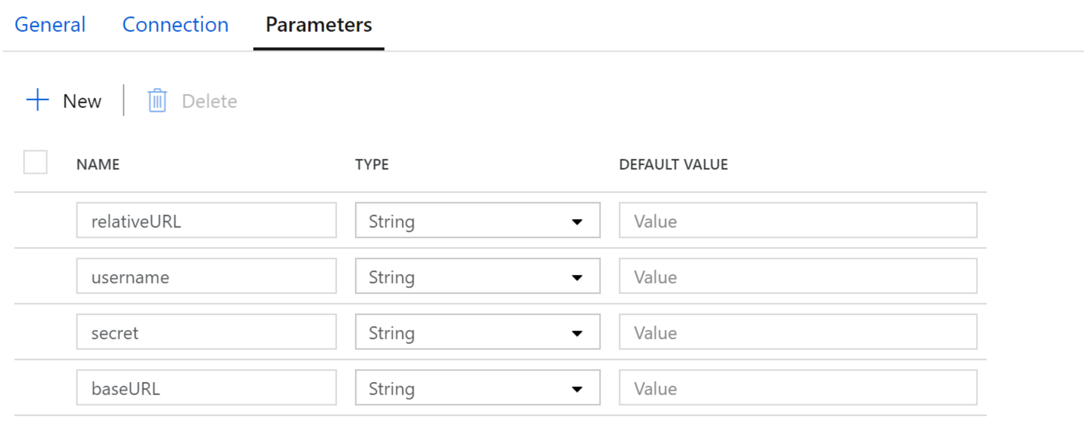


Pipeline with a parameterized copy activity

I have parameterized my linked service that points to the source of the data I am copying. My linked service has 3 parameters: BaseUrl, Username, and SecretName. The JSON for my linked service is below. You can see that I need to reference the parameter as the value for the appropriate property and also define the parameter at the bottom.

```
{
    "name": "LS_RESTSourceParam",
    "properties": {
        "annotations": [],
        "type": "RestService",
        "typeProperties": {
            "url": "@{linkedService().BaseUrl}",
            "enableServerCertificateValidation": true,
            "authenticationType": "Basic",
            "userName": "@{linkedService().Username}",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "MyKeyVault",
                    "type": "LinkedServiceReference"
                },
            "secretName": "@{linkedService().SecretName}"
            }
        },
        "parameters": {
            "Username": {
                "type": "String"
            },
            "SecretName": {
                "type": "String"
            },
            "BaseUrl": {
                "type": "String"
            }
        }
    }
}
```

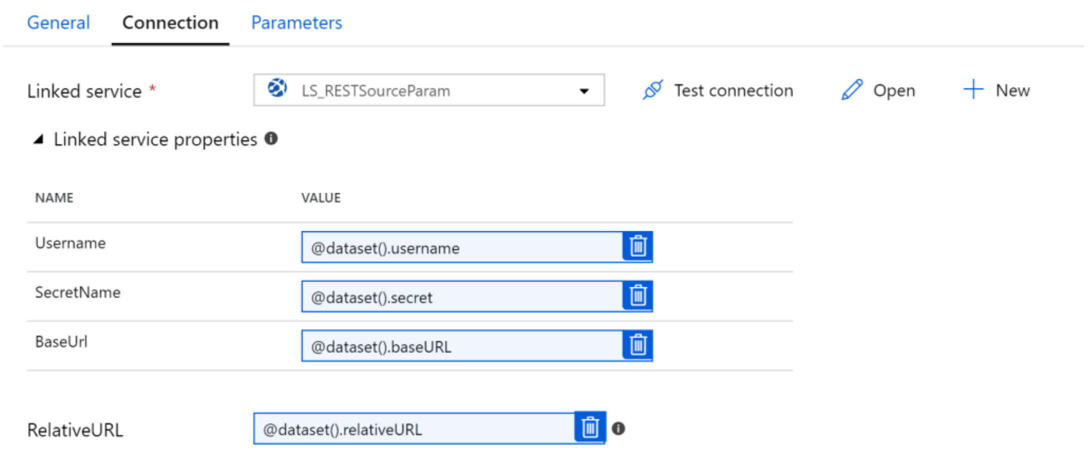
I have defined these three parameters in my dataset, along with one more parameter that is specific to the dataset (that doesn't get passed to the linked service). I don't need to set the default value on the Parameters tab of the dataset.



NAME	TYPE	DEFAULT VALUE
relativeURL	String	Value
username	String	Value
secret	String	Value
baseURL	String	Value

Parameters defined in the dataset

On the Connection tab of the dataset, I set the value as shown below. We can see that Data Factory recognizes that I have 3 parameters on the linked service being used. The relativeURL is only used in the dataset and is not used in the linked service. The value of each of these properties must match the parameter name on the Parameters tab of the dataset.



Linked service * LS_RESTSourceParam

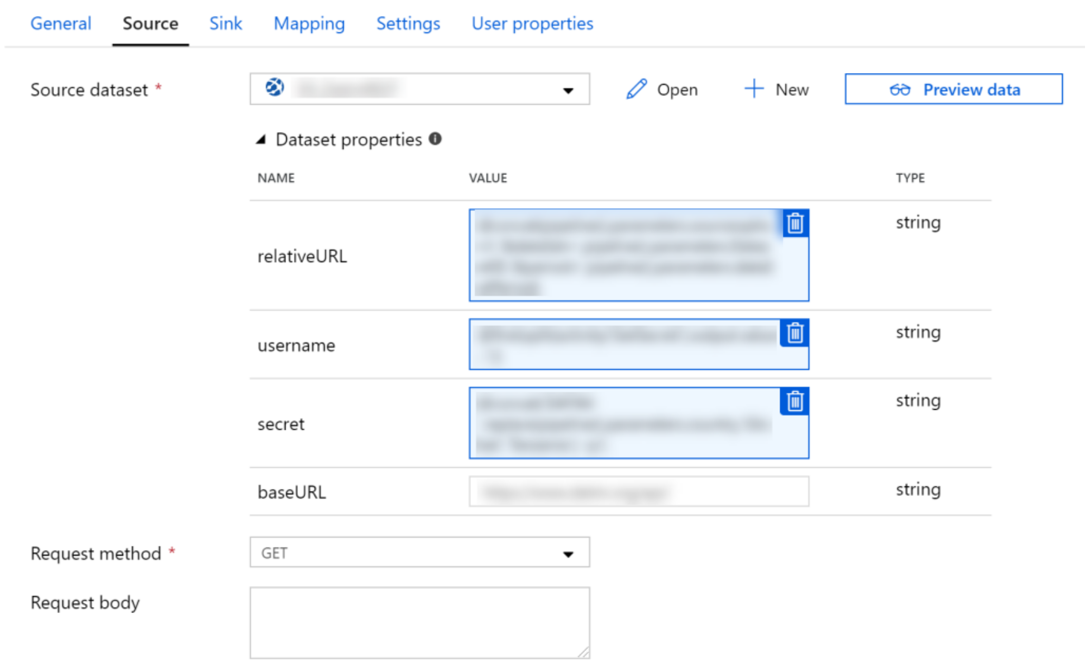
Linked service properties

NAME	VALUE
Username	@dataset().username
SecretName	@dataset().secret
BaseUrl	@dataset().baseURL

RelativeURL @dataset().relativeURL

Setting the properties on the Connection tab of the dataset

In my copy activity, I can see my 4 dataset parameters on the Source tab. There, I can write expressions to provide the values that should be passed through to the dataset, 3 of which are passed through to the linked service. In my case, this is a child pipeline that is called from a parent pipeline that passes in some values through pipeline parameters which are used in the expressions in the copy activity source.


General | Source | Sink | Mapping | Settings | User properties

Source dataset * [redacted] Open | New | Preview data

Dataset properties

NAME	VALUE	TYPE
relativeURL	[redacted]	string
username	[redacted]	string
secret	[redacted]	string
baseURL	[redacted]	string

Request method * GET

Request body

Defining the expressions for the dataset properties on the copy activity source

And that's it. I can run my pipeline and have it call different REST APIs using one linked service and one dataset.

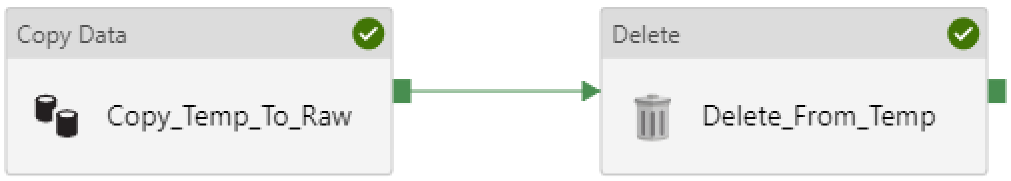
5 Comments
AZURE, **AZURE DATA FACTORY**, **MICROSOFT TECHNOLOGIES**

There is Now A Delete Activity in Data Factory V2!

March 7, 2019March 6, 2019 Meagan Longoria
Data Factory can be a great tool for cloud and hybrid data integration. But since its inception, it was less than straightforward how we should move data (https://datasavvy.me/2017/04/11/i-like-to-move-it-move-it-but-azure-data-factory-doesnt/) (copy to another location and delete the original copy).

It is a common practice (https://azure.microsoft.com/en-us/solutions/architecture/modern-data-warehouse/) to load data to blob storage or data lake storage before loading to a database, especially if your data is coming from outside of Azure. We often create a staging area in our data lakes (https://www.sqlchick.com/entries/2017/12/30/zones-in-a-data-lake) to hold data until it has been loaded to its next destination. Then we delete the data in the staging area once our subsequent load is successful. But before February 2019, there was no Delete activity. We had to write an Azure Function or use a Logic App called by a Web Activity (https://kromerbigdata.com/2018/03/15/azure-data-factory-delete-from-azure-blob-storage-and-table-storage/) in order to delete a file. I imagine every person who started working with Data Factory had to go and look this up.

But now Data Factory V2 has a Delete activity.



Data Factory V2 Copy activity followed by a Delete activity

How the Delete Activity Works

The Delete activity can delete from the following data stores:

- Azure blob storage
- ADLS Gen 1
- ADLS Gen 2
- File systems
- FTP
- SFTP
- Amazon S3

You can delete files or folders. You can also specifiy whether you want to delete recursively (delete including all subfolders of the specified folder). If you want to delete files/folders from a file system on a private network (e.g., on premises), you will need to use a self-hosted integration runtime running version 3.14 or higher. Data Factory will need write access to your data store in order to perform the delete.

You can log the deleted file names as part of the Delete activity. It requires you to provide a blob storage or ADLS Gen 1 or 2 account as a place to write the logs.

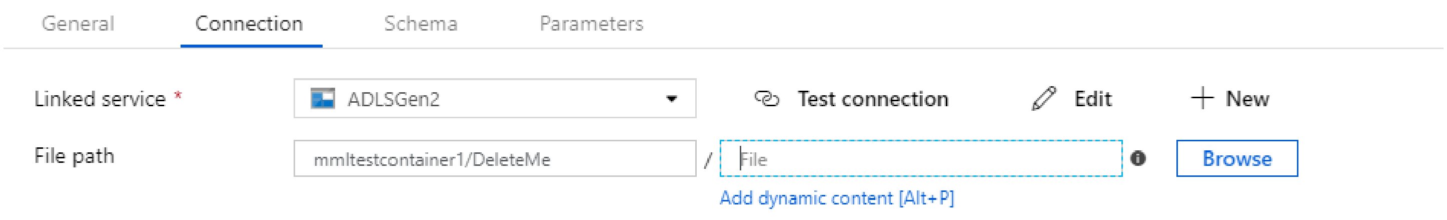
You can parameterize the following properties in the Delete activity itself:

- Timeout
- Retry
- Delete file recursively
- Max concurrent connections
- Enable Logging
- Logging folder path

You can also parameterize your dataset as usual.

All that's required in the Delete activity is an activity name and dataset. The other properties are optional. Just be sure you have specified the appropriate file path. Maybe try this out in dev before you accidentally delete your way through prod.

To delete all contents of a folder (including subfolders), specify the folder path in your dataset and leave the file name blank, then check the box for "Delete file recursively".



Linked service * ADLSGen2
File path mmltestcontainer1/DeleteMe / File

You can use a wildcard (*) to specify files, but it cannot be used for folders (https://docs.microsoft.com/en-us/azure/data-factory/delete-activity#known-limitation).

Here's to much more efficient development of data movement pipelines in Azure Data Factory in V2.

4 Comments
AZURE, **AZURE DATA FACTORY**, **MICROSOFT TECHNOLOGIES**, **POWER BI**

How Many Data Gateways Does My Azure BI Architecture Need?

February 7, 2019September 3, 2019 Meagan Longoria

It's not always obvious when you need a data gateway in Azure, and not all gateways are labeled as such. So I thought I would walk through various applications that act as a data gateway and discuss when, where, and how many are needed.

Note: I'm ignoring VPN gateways and application gateways for the rest of this post. I'm assuming your networking/VPN situation is fixed at this point and working from there.

Let's start with what services may require you to use a data gateway.

You will need a data gateway when you are using Power BI, Azure Analysis Services, PowerApps, Microsoft Flow, Azure Logic Apps, Azure Data Factory, or Azure ML with a data source/destination that is in a private network. Note that a private network includes on-premises data sources and Azure Virtual Machines as well as Azure SQL Databases and Azure SQL Data Warehouses that require use of VNet service endpoints rather than public endpoints.

Luckily, many of these services can use the same data gateway. Power BI, Azure Analysis Services, PowerApps, Microsoft Flow, and Logic Apps all use the On Premises Data Gateway (https://docs.microsoft.com/en-us/power-bi/service-gateway-onprem-indepth). Azure Data Factory

(V1 and V2) and Azure Machine Learning Studio use the Data Factory Self-Hosted Integration Runtime (https://docs.microsoft.com/en-us/azure/data-factory/concepts-integration-runtime#self-hosted-integration-runtime).

On Premises Data Gateway (Power BI et al.)

If you are using one or more of the following:

- Power BI
- Azure Analysis Services
- PowerApps
- Microsoft Flow
- Logic Apps

and you have a data source in a private network, you need at least one gateway. But there are a few considerations that might cause you to set up more gateways.

1. Your services must be in the same region to use the same gateway. This means that your Power BI/Office 365 region and Azure region for your Azure Analysis Services resource must match (https://youtu.be/aywlluB9FHo) for them to all use one gateway. If you have resources in different regions, you will need one gateway per region.
2. You may want high availability for your gateway. You can create high availability clusters (https://docs.microsoft.com/en-us/power-bi/service-gateway-high-availability-clusters) so when one gateway is down, traffic is rerouted to another available gateway in the cluster.
3. You may want to segment traffic to ensure the necessary resources for certain ad hoc live/direct queries or scheduled refreshes. If your usage and refresh patterns warrant it, you may want to set up one gateway for scheduled refreshes and one gateway for live/direct queries back to any on-premises data sources. Or you might make sure live/direct queries for two different high-traffic models go through different gateways so as not to block each other. This isn't always warranted, but it can be a good strategy.

Data Factory Self-hosted Integration Runtime

If you are using Azure Data Factory (V1 or V2) or Azure ML with a data source in a private network, you will need at least one gateway. But that gateway is called a Self-hosted Integration Runtime (IR).

Self-hosted IRs can be shared across data factories (https://azure.microsoft.com/en-us/blog/sharing-a-self-hosted-integration-runtime-infrastructure-with-multiple-data-factories/) in the same Azure Active Directory tenant. They can be associated with up to four machines to scale out or provide higher availability. So while you may only need one node, you might want a second so that your IR is not the single point of failure.

Or you may want multiple IRs to boost throughput of copy activities. For instance, copying from an on-premises file server with one IR node is about 195 Megabytes per second (MB/s) (https://docs.microsoft.com/en-us/azure/data-factory/copy-activity-performance). But with 4 IR nodes, it can be as fast as 505 MB/s (https://docs.microsoft.com/en-us/azure/data-factory/copy-activity-performance).

Factors that Affect the Number of Data Gateways Needed

The main factors determining the number of gateways you need are:

1. Number of data sources in private networks (including Azure VNets)
2. Location of services in Azure and O365 (number of regions and tenants)
3. Desire for high availability
4. Desire for increased throughput or segmented traffic

If you are importing your data to Azure and using an Azure SQL DB with no VNet as the source for your Power BI model, you won't need an On Premises Data Gateway. If you used Data Factory to copy your data from an on-premises SQL Server to Azure Data Lake and then Azure SQL DB, you need a Self-Hosted Integration Runtime.

If all your source data is already in Azure, and your source for Power BI or Azure Analysis Services is Azure SQL DW on a VNet, you will need at least one On-Premises Data Gateway.

If you import a lot of data to Azure every day using Data Factory, and you land that data to Azure SQL DW on a VNet, then use Azure Analysis Services as the data source for Power BI reports, you might want a self-hosted integration runtime with a few nodes and a couple of on-premises gateways clustered for high availability.

Have a Plan For Your Gateways

The gateways/integration runtimes are not hard to install. They are just often not considered, and projects get stalled waiting until a machine is provisioned to install them on. And many people forget to plan for high availability in their gateways. Make sure you have the right number of gateways and IR nodes to get your desired features and connectivity. You can add gateways/nodes later, but you don't want to get caught with no high availability when it really matters.

8 Comments
AZURE, **AZURE DATA FACTORY**, **MICROSOFT TECHNOLOGIES**

Data Factory V2 Activity Dependencies are a Logical AND

October 2, 2018October 3, 2018 Meagan Longoria
Azure Data Factory V2 allows developers to branch and chain activities together (https://docs.microsoft.com/en-us/azure/data-factory/tutorial-control-flow) in a pipeline. We define dependencies between activities as well as their their dependency conditions. Dependency conditions can be succeeded, failed, skipped, or completed.

This sounds similar to SSIS precedence constraints (https://www.mssqltips.com/sqlservertip/4761/defining-workflow-in-ssis-using-precedence-constraints/), but there are a couple of big differences.

1. SSIS allows us to define expressions to be evaluated to determine if the next task should be executed.

2. SSIS allows us to choose whether we handle multiple constraints as a logical AND or a logical OR. In other words, do we need all constraints to be true or just one.

ADF V2 activity dependencies are always a logical AND. While we can design control flows in ADF similar to how we might design control flows in SSIS, this is one of several differences. Let's look at an example.

Data Factory V2 Pipeline with no failure dependencies

The pipeline above is a fairly common pattern. In addition to the normal ADF monitoring that is available with the product, we may log additional information to a database or file. That is what is happening in the first activity, logging the start of the pipeline execution to a database table via a stored procedure.

The second activity is a Lookup that gets a list of tables that should be loaded from a source system to a data lake. The next activity is a ForEach, executing the specified child activities for each value passed along from the list returned by the lookup. In this case the child activity includes copying data from a source to a file in the data lake.

Finally, we log the end of the pipeline execution to the database table.

Activities on Failure

This is all great as long as everything works. What if we want something else to happen in the event that one of the middle two activities fail?

This is where activity dependencies come in. Let's say I have a stored procedure that I want to run when the Lookup or ForEach activity fails. Your first instinct might be to do the below.

Data Factory V2 Pipeline with two dependencies on failure activity

The above control flow probably won't serve you very well. The LogFailure activity will not execute unless both the Lookup activity and the ForEach activity fails. There is no way to change the dependency condition so that LogFailure executes if the Lookup OR the ForEach fails.

Instead, you have a few options:

1). Use multiple failure activities.

Pipeline with stored procedure executed when the Lookup or ForEach activity fails

This is probably the most straight forward but least elegant option. In this option you add one activity for each potential point of failure. The stored procedure you execute in the LogLookupFailure and LogForEachFailure activities may be the same, but you need the activities to be separate so there is only one dependency for execution.

2) Create a parent pipeline and use an execute pipeline activity. Then add a single failure dependency from a stored procedure to the execute pipeline activity. This works best if you don't really care in which activity your original/child pipeline failed and just want to log that it failed.

Execute pipeline activity with a stored procedure executed on failure

3) Use an If Condition (https://docs.microsoft.com/en-us/azure/data-factory/control-flow-if-condition-activity) activity and write an expression that would tell you that your previous activity failed. In my specific case I might set some activity dependencies to completed instead of success and replace the LogPipelineEnd stored procedure activity with the If Condition activity. If we choose a condition that indicates failure, our If True activity would execute the failure stored procedure and our If False activity would execute the success stored procedure.

Think of it as a dependency, not a precedence constraint.

It's probably better to think of activity dependencies as being different than precedence constraints. This becomes even more obvious if we look at the JSON that we would write to define this rather than using the GUI. MyActivity2 depends on MyActivity1 succeeding. If we add another dependency in MyActivity2, it would depend both on that new one and the original dependency. Each additional dependency is added on.

```
{
    "name": "MyPipeline",
    "properties":
    {
        "description": "pipeline description",
        "activities": [
         {
            "name": "MyActivity1",
            "type": "Copy",
            "typeProperties": {
            },
            "linkedServiceName": {
            }
        },
        {
            "name": "MyActivity2",
            "type": "Copy",
            "typeProperties": {
            },
            "linkedServiceName": {
            },
            "dependsOn": [
            {
                "activity": "MyActivity1",
                "dependencyConditions": [
                    "Succeeded"
                ]
            }
          ]
        }
      ],
      "parameters": {
       }
    }
}
```

Do you have another way of handling this in Data Factory V2? Let me know in the comments.

If you would like to see Data Factory V2 change to let you choose how to handle multiple dependencies, you can vote for this idea (https://feedback.azure.com/forums/270578-data-factory/suggestions/35616982-allow-choosing-logical-and-or-logical-or-in-activi) on the Azure feedback site or log your own idea to suggest a different enhancement to better handle this in ADF V2.

10 Comments

AZURE DATA FACTORY, BIML, CONFERENCES, SSIS

I'm Speaking at IT/Dev Connections 2017

May 24, 2017 Meagan Longoria
I'm pleased to say that I am speaking at IT/Dev Connections 2017. This year the conference will be held in San Francisco October 23-26. I had a great experience speaking at IT/Dev Connections in 2015, so I am excited to return again this year.

This conference is special to me because of its focus on providing great content for developers and IT pros – the conference website describes it as the "anti-keynote" conference (http://www.devconnections.com/dc17/Public/Content.aspx?ID=1068156&sortMenu=102002) with no forced marketing content.

I also enjoy it because it is more than just SQL Server/Data Platform (they have tracks for Cloud & Data Center, Enterprise Collaboration, Development & Dev Ops, and Enterprise Mobility and Security), and it's nice to get out of my comfort zone a bit.

I will deliver two sessions (http://www.devconnections.com/dc17/Public/SpeakerDetails.aspx?nav=true&FromPage=Speakers.aspx&ContactID=1178374) at the conference.

Azure Data Factory in A Nutshell

If you have been wanting to get into Azure Data Factory (ADF) development, join me for this demo-filled overview. In this session, we'll go over the basic anatomy of an ADF solution. You'll learn what ADF is and isn't as we walk through a solution to pull data from an on-premises SQL Server database to a blob storage and then populate and Azure SQL Data Warehouse. You'll learn tips for creating ADF solutions in Visual Studio, and I'll show you how to make ADF development less tedious with a free Visual Studio Add-in called Biml Express. You'll leave with a basic understanding of ADF and a list of tools and skills you'll want to acquire as you begin your ADF development.

Improve Data Warehouse ETL Delivery with a Patterns-Based Approach

What if I told you that 90% of your data integration development in SQL Server could be automated? In 5 years, you will be "old fashioned" if you are hand coding SSIS packages. Developers with different skill levels and design preferences create databases and SSIS packages however they see fit to get the job done. Documentation is frequently omitted. Maintenance and small enhancements consume too much development time while manual errors and inconsistencies slip through the testing and release process. You can use tools and frameworks to rearrange the development process and alleviate these common problems. The implementation and automation of design patterns leads to improved efficiency and communication. Join me in this session to learn how to use Business Intelligence Markup Language (Biml) and Excel to facilitate metadata-driven SSIS development. I'll use database schema information plus Excel inputs to implement a small data mart from staging through the dimensional model.

I hope you will join me in San Francisco in October!

2 Comments
AZURE, **AZURE DATA FACTORY**, **MICROSOFT TECHNOLOGIES**

Azure Data Factory and the Case of the Missing JRE That Wasn't

May 1, 2017March 6, 2019 Meagan Longoria

Note: This post was written about Azure Data Factory V1, but is also applicable to V2.

On a recent project I used Azure Data Factory (ADF) to retrieve data from an on premises SQL Server 2014 instance and land them in Azure Data Lake Store (ADLS) as ORC files (https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC). This required the use of the Data Management Gateway (https://docs.microsoft.com/en-us/azure/data-factory/data-factory-data-management-gateway) (DMG). Setup was quick and easy in our development environment. We installed the DMG for development on a separate server in the client's network, where we also installed SQL Server Management Studio (SSMS) for query development and data validation. We set up resource groups in Azure for development and production, and made sure the settings for development and production were the same. Then we set up a separate server for the production DMG.

Deployment and execution went well in the dev environment. Testing was completed, so we deployed to prod. Deployment went fine, but the pipelines failed execution and returned the following error on the output data sets.

Java Runtime Environment is not found.

The Java Runtime Environment (JRE) is not required for the DMG to run successfully, but it is needed for the creation of ORC files. The only problem with this error message was that we did indeed have the JRE installed on the server.

After reinstalling the JRE and the DMG and getting the same error, I consulted the troubleshooting (https://docs.microsoft.com/en-us/azure/data-factory/data-factory-troubleshoot-gateway-issues) guide. After finding nothing relevant there, I asked some colleagues for suggestions.

- I double-checked that I had the same version of the DMG that worked in dev and that I had the most current version of the JRE.
- I double-checked that the DMG and JRE matched bit-wise (32-bit vs. 64-bit). Both were 64-bit in my case.
- I checked that JAVA_HOME was set correctly in the environment variables.

When none of those things worked, I logged a support ticket with Microsoft. They had me do the following:

- Check the registry key – HKEY_LOCAL_MACHINE\Software\JavaSoft\Java Runtime Environment should have a Current Version entry that shows the current JRE version.
- Check that the subkey in the folder labeled with the version has a JAVAHOME entry with the correct path (something like C:\Program Files\Java\jre1.8.0_74).
- Open the path and check that the bin folder exists.
- Check that jvm.dll exists in the bin/server folder.

When none of those things worked, they gave me one last suggestion:

Install the Microsoft Visual C++ 2010 Redistributable Package (https://www.microsoft.com/en-us/download/details.aspx?id=14632).

And that turned out to solve the problem!

After review, we realized that we had installed SSMS on the dev DMG server but not on the prod DMG server. SSMS would have required the installation of the C++ redistributable package, which is why we didn't encounter this error in dev.

I will confess that I don't understand exactly why missing C++ libraries manifest themselves in an error claiming a missing Java Runtime Environment. If you have a good explanation, please leave it in the comments and I'll update this and give you credit.

I hope that someone else who runs into this issue will find this blog post and avoid days of troubleshooting and confusion.

7 Comments
AZURE, AZURE DATA FACTORY, AZURE DATA LAKE, MICROSOFT TECHNOLOGIES

I Like to Move It, Move It – But Azure Data Factory Doesn't

April 11, 2017March 6, 2019 Meagan Longoria
Note: This post is about Azure Data Factory V1
I've spent the last couple of months working on a project that includes Azure Data Factory and Azure Data Warehouse. ADF has some nice capabilities for file management that never made it into SSIS such as zip/unzip files and copy from/to SFTP. But it also has some gaps I had to work around. My

project involved copying data from on-premises SQL Server to an ORC file in a data lake staging area for ingestion into an Azure SQL Data Warehouse through Polybase. Then I had planned to move that file to the a raw area of the data lake for archiving.

In other words, as sung below by a great lemur, I like to move it.

But at this time ADF doesn't support that. You can copy a file with a copy activity, but you cannot actually move (i.e., copy and delete).

Luckily, we had a workaround for our situation. If you tell ADF to copy data to a file that already exists in the specified location in the data lake, it will overwrite the existing file. We made sure the file name is always the same for each table in the staging area so there is always only one file per table.

What we ultimately ended up with was:

1. Retrieve time sliced data from on-premises SQL Server source via the Data Management Gateway.
2. Land data in the Raw area of the data lake as ORC file.
3. Copy file to staging.
4. Execute stored procedure to populate data warehouse through Polybase.

I landed the data in Raw first so that we would not have to pull from SQL again if we needed to re-run a slice. Data latency wasn't a huge issue for this client – we had some pipelines that ran hourly and some that ran daily. The extra seconds it took to land the file in Raw was not a concern.

For now, if you do need to actually move or delete, you can use a custom C# activity (http://stackoverflow.com/questions/41735506/moving-not-copying-data-in-azure-data-factory) to delete files. I chose not to do this because I didn't want to add another technology for the client to learn/manage while adopting Azure. This may be the way to go for other projects.

If you think moving (copying and deleting) files should be a first class citizen in Azure Data Factory, please vote for the idea and spread the word for others to vote.

Meagan Longoria
@MMarie

I support 'Move Activity' - An activity that copies and then deletes in Azure Data Factory. What do you think?

Move Activity
feedback.azure.com

10:40 PM · Apr 2, 2017

3 See Meagan Longoria's other Tweets

You don't have to thank me for getting that song stuck in your head for the rest of the day.

1 Comment
AZURE, **AZURE DATA FACTORY**, **AZURE DATA LAKE**, **BIML**, **MICROSOFT TECHNOLOGIES**

Copying data from On Prem SQL to ADLS with ADF and Biml – Part 2

March 10, 2017March 6, 2019 Meagan Longoria
Note: This post is about Azure Data Factory V1
I showed in my previous post (https://datasavvy.me/2017/03/03/copying-data-from-on-prem-sql-to-adls-with-adf-and-biml-part-1/) how we generated the datasets for our Azure Data Factory pipelines. In this post, I'll show the BimlScript for our pipelines. Pipelines define the activities, identify the input and output datasets for those activities, and set an execution schedule. We were creating several pipelines with copy activities to copy data to Azure Data Lake Store.

We generated one pipeline per schedule and load type:

- Hourly – Full

- Hourly – Incremental
- Daily – Full
- Daily – Incremental

We also generated some one-time load pipelines for DR/new environment setup.

The first code file below is the template for the pipeline. You can see code nuggets for the data we receive from the generator file and for conditional logic we implemented. The result is one copy activity per source table within the appropriate pipeline.

In the second code file below, lines 104 to 119 are generating the pipelines. We read in the necessary data from the Excel file:

- Schema name
- Table name
- Columns list
- Incremental predicate

Sidenote: We wrote a quick T-SQL statement (not shown) to generate the columns list. This could have been done in our BimlScript, but it was something we changed after the fact to accommodate the limitations of Polybase (https://datasavvy.me/2016/08/01/polybase-is-a-picky-eater-remove-carriage-returns-before-ingesting-text/) (Dear Microsoft: Please fix (https://feedback.azure.com/forums/307516-sql-data-warehouse/suggestions/10600132-polybase-allow-field-row-terminators-within-strin)). SQL was quicker and easier for us, but if I were to do this again I would add that into our BimlScript. We needed to replace new lines and double quotes in our data before we could read it in from the data lake. You can get around this issue by using .ORC files rather than text delimited files. But the ORC files aren't human readable, and we felt that was important for adoption of the data lake with the client on this project. They were already jumping in with several new technologies and we didn't want to add anything else to the stack. So our select statements list out fields and replace the unwanted characters in the string fields.

Our Excel file looks like this.

Columns B, C, L, and M are populated by Excel formulas. This is the file that is read in by the BimlScript in the code below.

In our generator file (which is the same file that was used to generate the datasets), we use the CallBimlScript function to call the pipeline template file and pass along the required properties (table, schema, frequency, scope, columns list, predicate).

```
<#@ import namespace="System.Data" #>
<#@ import namespace="System.Text" #>
<#@ property name="targetTables" type="DataView"#>
<#@ property name="frequency" type="string"#>
<#@ property name="scope" type="string"#>
{
    "$schema": "http://datafactories.schema.management.azure.com/schemas/2015-09-01/Microsc
    "name": "PL_Copy_MySourceDBToADLS_<#=frequency#>_<#=scope#>",
  "properties": {
    "description": "<#=frequency#> <#=scope#> copies of data from Source db to the data lake.",
    "activities": [
    <# var isFirst = true; foreach( DataRowView rowView in targetTables) {#>
    <# DataRow row = rowView.Row; #>
    <# string schemaName = row["SchemaName"].ToString();#>
    <# string tableName = row["TableName"].ToString();#>
    <# string columnList = row["ColumnListForSelect"].ToString(); #>
    <# string predicate = row["IncrementalPredicate"].ToString(); #>
    <#=isFirst ? "" : ","#>
      {
        "name": "Copy to Lake – <#=schemaName#>.<#=tableName#>",
        "type": "Copy",
        "inputs": [
          {
            "name": "DS_OnPremSQL_MySourceDB_<#=schemaName#>_<#=tableName#>"
          }
        ],
        "outputs": [
          {
            "name": "DS_DataLake_MySourceDB_<#=schemaName#>_<#=tableName#>"
          }
        ],
        "typeProperties": {
          "source": {
            "type": "SqlSource",
            <# if (scope == "Full") {#>
            "sqlReaderQuery": "SELECT <#=columnList#>, SYSDATETIME() AS LoadDateTime FRO
            <#} else if (scope == "Deltas" && frequency == "Hourly") {#>
            "sqlReaderQuery": "$$Text.Format('SELECT <#=columnList#>, SYSDATETIME() AS Loac
            <#} else if (scope == "Deltas" && frequency == "Daily") {#>
            "sqlReaderQuery": "$$Text.Format('SELECT <#=columnList#>, SYSDATETIME() AS Loac
            <# } #>
          },

          "sink": {
            "type": "AzureDataLakeStoreSink"
          }
        },
        "policy": {
          "concurrency": 1,
          "executionPriorityOrder": "OldestFirst",
          "retry": 3,
          "timeout": "01:00:00"
        },
        "scheduler": {
        <# if (frequency == "Daily") {#>
            "frequency": "Day",
            "offset": "09:00:00",
            <#} else if (frequency == "Hourly") {#>
            "frequency": "Hour",
            <# } #>
          "style": "EndOfInterval",
          "interval": 1
        }
      }
      <# isFirst = false; }#>
    ],
    <# if (frequency == "Hourly") {#>
    "start": "2017-03-01T01:00:00",
    <#}else {#>
    "start": "2017-03-02T00:00:00",
    <#}#>
    "end": "9999-09-09"
  }
}
```

view raw
PL_Copy_MySourceDBToADLS.biml
hosted with ❤ by **GitHub**

```
<#@ template tier="10" #>
<#@ import namespace="System.Data" #>
<#@ import namespace="System.Text" #>
<#@ code file="BGHelper.cs" #>
<#@ import namespace="BGHelper" #>

<Biml xmlns="http://schemas.varigence.com/biml.xsd"&gt (http://schemas.varigence.com/biml
</Biml>
<#
    string mdFilePath = "C:\\Users\\admin\\Source\\Workspaces\\Data Warehouse\\meta
    string mdFileName = "TargetTableMetadata.xlsx";
    string mdWorkSheetName = "Metadata$";
    bool mdHasHeader = true;

    string logPath = "C:\\Users\\admin\\Source\\Workspaces\\Data Warehouse\\data_fac
    string adfProjPath = "C:\\Users\\admin\\Source\\Workspaces\\Data Warehouse\\data

    DataSet ds = new DataSet();

```

```
ds = ExcelReader.ReadExcelQuery(mdFilePath, mdFileName, mdWorkSheetName, mdHas

System.IO.File.AppendAllText(@logPath, "MetaData File Path: " + System.IO.Path.Combine(
System.IO.File.AppendAllText(@logPath, "MetaData File Path: " + System.IO.File.Exists(Syst
System.IO.File.AppendAllText(@logPath, "Dataset table count: " + ds.Tables.Count.ToString(

DataView dailyFulls = new DataView(ds.Tables["Metadata"],"Frequency = 'Daily' and [Chan

DataView dailyDeltas = new DataView(ds.Tables["Metadata"], "Frequency = 'Daily' and [Ch

DataView hourlyFulls = new DataView(ds.Tables["Metadata"], "Frequency = 'Hourly' and [C

DataView hourlyDeltas = new DataView(ds.Tables["Metadata"], "Frequency = 'Hourly' and [


//log count of results for each filter
System.IO.File.AppendAllText(@logPath, "Daily Fulls Count: " + dailyFulls.Count.ToString()
System.IO.File.AppendAllText(@logPath, "Daily Deltas Count: " + dailyDeltas.Count.ToStrin
System.IO.File.AppendAllText(@logPath, "Hourly Fulls Count: " + hourlyFulls.Count.ToStri
System.IO.File.AppendAllText(@logPath, "Hourly Deltas Count: " + hourlyDeltas.Count.ToS

//Generate datasets

foreach (DataRowView rowView in dailyFulls)
{
    DataRow row = rowView.Row;

    string schemaName = row["SchemaName"].ToString();
    string tableName = row["TableName"].ToString();
    string frequency = row["Frequency"].ToString();
    string scope = "full";

    System.IO.File.AppendAllText(@logPath, "DailyFulls | "  + row["SchemaName"].ToString(

    System.IO.File.WriteAllText(@adfProjPath + "DS_OnPremSQL_MySourceDB_" + schemaN
    System.IO.File.WriteAllText(@adfProjPath + "DS_DataLake_MySourceDB_" + schemaNam
}

foreach (DataRowView rowView in dailyDeltas)
{
    DataRow row = rowView.Row;

    string schemaName = row["SchemaName"].ToString();
    string tableName = row["TableName"].ToString();
    string frequency = row["Frequency"].ToString();
```

```
        string scope = "deltas";

        System.IO.File.AppendAllText(@logPath, "DailyFulls | "  + row["SchemaName"].ToString(

        System.IO.File.WriteAllText(@adfProjPath + "DS_OnPremSQL_MySourceDB_" + schemaN
        System.IO.File.WriteAllText(@adfProjPath + "DS_DataLake_MySourceDB_" + schemaNam
    }

    foreach (DataRowView rowView in hourlyFulls)
    {
        DataRow row = rowView.Row;

        string schemaName = row["SchemaName"].ToString();
        string tableName = row["TableName"].ToString();
        string frequency = row["Frequency"].ToString();
        string scope = "full";

        System.IO.File.AppendAllText(@logPath, "DailyFulls | "  + row["SchemaName"].ToString(

        System.IO.File.WriteAllText(@adfProjPath + "DS_OnPremSQL_MySourceDB_" + schemaN
        System.IO.File.WriteAllText(@adfProjPath + "DS_DataLake_MySourceDB_" + schemaNam
    }

    foreach (DataRowView rowView in hourlyDeltas)
    {
        DataRow row = rowView.Row;

        string schemaName = row["SchemaName"].ToString();
        string tableName = row["TableName"].ToString();
        string frequency = row["Frequency"].ToString();
        string scope = "deltas";

        System.IO.File.AppendAllText(@logPath, "DailyFulls | "  + row["SchemaName"].ToString(

        System.IO.File.WriteAllText(@adfProjPath + "DS_OnPremSQL_MySourceDB_" + schemaN
        System.IO.File.WriteAllText(@adfProjPath + "DS_DataLake_MySourceDB_" + schemaNam
    }


    // Generate pipelines
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_Daily_Full.js
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_Daily_Deltas
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_Hourly_Full.
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_Hourly_Delt

    //Generate One-Time Pipelines
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_OneTime_Da
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_OneTime_Da
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_OneTime_Ho
    System.IO.File.WriteAllText(@adfProjPath + "PL_Copy_MySourceDBToADLS_OneTime_Ho




    #>
```

view raw
ADFGenerator.biml
hosted with ❤ by **GitHub**

The great thing about Biml is that I can use it as much or as little as I feel is helpful. That T-SQL statement to get column lists could have been Biml, but it didn't have to be. The client can maintain and enhance these pipelines with or without Biml as they see fit. There is no vendor lock-in here. Just as with Biml-generated SSIS projects, there is no difference between a hand-written ADF solution and a Biml-generated ADF solution, other than the Biml-generated solution is probably more consistent.

And have I mentioned the time savings? There is a reason why Varigence gives out shirts that say "It's Monday and I'm done for the week."

Raymond Sondak
@raymondsondak

1 of my fav #Biml quote "It's Monday, and I'M DONE for the week!" -Biml User-. Yes thats you in the pic @adenhaan :)

We made changes and regenerated our pipelines a few times, which would have taken hours without Biml. With Biml, it was no big deal.

Thanks to Levi (https://twitter.com/levisyck) for letting me share some of his code, and for working with me on this project!

1 Comment

Data Savvy

Blog at WordPress.com.