Best Practices for Implementing Azure Data Factory

Posted on December 18, 2019August 23, 2021
Blog post updated 14th August 2021 🙂

(https://mrpaulandrew.files.wordpress.com/2019/12/1st-prize-adf-update.png)My colleagues and friends from the community keep asking me the same thing… What are the best practices from using Azure Data Factory (ADF)? With any emerging, rapidly changing technology I'm always hesitant about the answer. However, after 6 years of working with ADF I think its time to start suggesting what I'd expect to see in any good Data Factory implementation, one that is running in production as part of a wider data platform solution. We can call this technical standards or best practices if you like. In either case, I would phrase the question; what makes a good Data Factory implementation?

The following are my suggested answers to this and what I think makes a good Data Factory. Hopefully together we can mature these things into a common set of best practices or industry standards when using the cloud resource. Some of the below statements might seem obvious and fairly simple. But maybe not to everybody, especially if your new to ADF.

Contents

Platform Setup	Other Stuff…
	Azure Data Factory by Example ISBN-13978-1484270288 I did the technical review (https://mrpaulandrew.com/2021/06/29/azure-data-factory-by-exampe-a-review-of-my-technical-review/) of

- Azure Integration Runtimes
- SSIS Integration Runtimes *(new)*

Parallelism

- Parallel Execution
- Service Limitations
- Understanding Activity Concurrency *(new)*

Reusable Code

- Dynamic Linked Services
- Generic Datasets
- Pipeline Hierarchies *(updated)*
- Metadata Driven Processing
- Using Templates

Security

- Linked Service Security via Azure Key Vault
- Getting Key Vault Secrets at Runtime *(new)*
- Managed Identities *(new)*
- Security Custom Roles
- Securing Activity Inputs & Outputs *(new)*

Monitoring & Error Handling

- Custom Error Handler Paths
- Monitoring via Log Analytics
- Timeouts & Retry

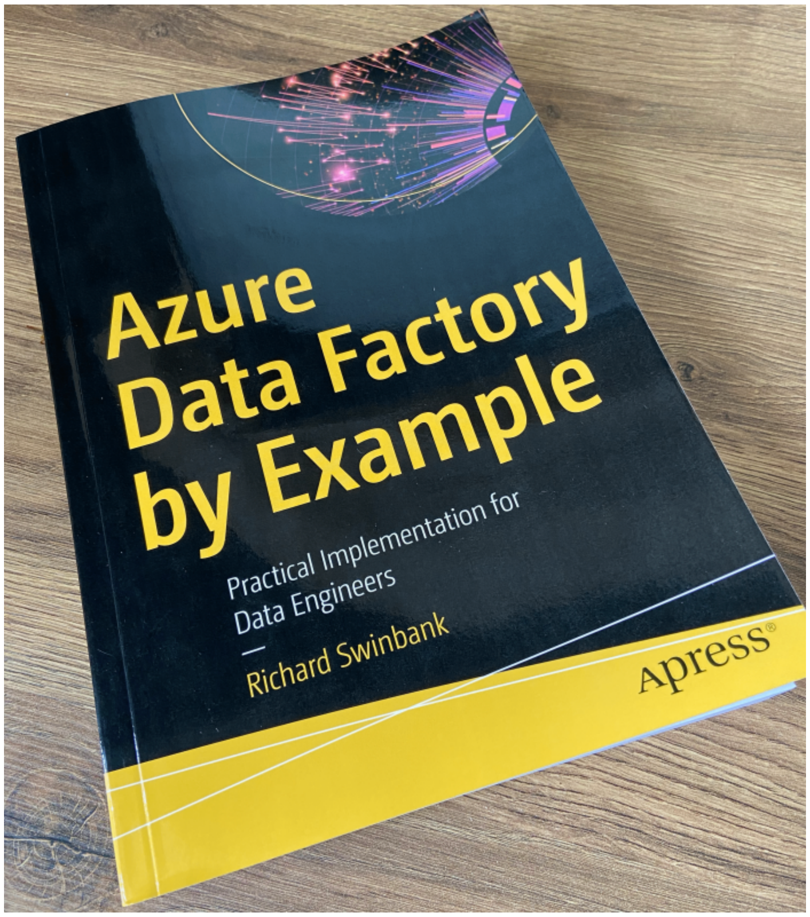
Cosmetic Stuff

- Naming Conventions
- Annotations
- Pipeline & Activity Descriptions
- Factory Component Folders

Documentation

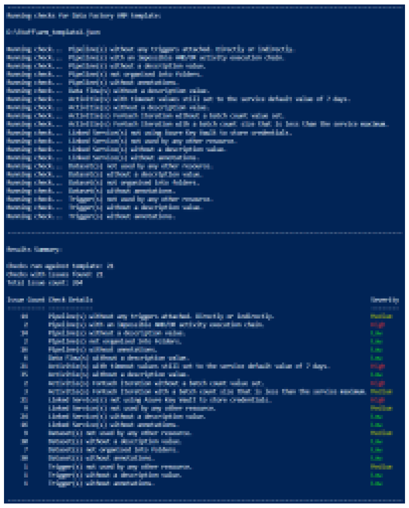
- Documentation *(updated)*
- Visio Stencils *(new)*

this book with Richard, it has a lot of great practical content. Check it out if you prefer a detailed guide on creating a good Data Factory.


Azure Data Factory by Example
Practical Implementation for Data Engineers
Richard Swinbank
Apress

PowerShell Checker Script

As a side project I recently created a PowerShell script to inspect an ARM template export for a given Data Factory instance and provide a summarised check list coverings a partial set of the bullet points on the right, view the blog post here (https://mrpaulandrew.com/2020/11/09/best-practices-for-implementing-azure-data-factory-auto-checker-script-v0-1/).



```
Running checks for Data Factory ARM template:

[illegible]

Running check... Pipeline(s) without any triggers attached. Directly or indirectly.
Running check... Pipeline(s) with an impossible AND/OR activity execution chain.
Running check... Pipeline(s) without a description value.
Running check... Pipeline(s) not organised into folders.
Running check... Pipeline(s) without annotations.
Running check... Data Flow(s) without a description value.
Running check... Activitie(s) with timeout values still set to the service default value of 7 days.
Running check... Activitie(s) without a description value.
Running check... Activitie(s) ForEach iteration without a batch count value set.
Running check... Activitie(s) ForEach iteration with a batch count size that is less than the service maximum.
Running check... Linked Service(s) not using Azure Key Vault to store credentials.
Running check... Linked Service(s) not used by any other resource.
Running check... Linked Service(s) without a description value.
Running check... Linked Service(s) without annotations.
Running check... Dataset(s) not used by any other resource.
Running check... Dataset(s) without a description value.
Running check... Dataset(s) not organised into folders.
Running check... Dataset(s) without annotations.
Running check... Trigger(s) not used by any other resource.
Running check... Trigger(s) without a description value.
Running check... Trigger(s) without annotations.

Results Summary:

Checks ran against template: [illegible]
Checks with issues found: [illegible]
Total issue count: [illegible]

Issue Count Check Details                                                                          Severity
----------- -------------                                                                          --------
 [illegible]   Pipeline(s) without any triggers attached. Directly or indirectly.                  Medium
 [illegible]   Pipeline(s) with an impossible AND/OR activity execution chain.                     High
 [illegible]   Pipeline(s) without a description value.                                            Low
 [illegible]   Pipeline(s) not organised into folders.                                             Low
 [illegible]   Pipeline(s) without annotations.                                                    Low
 [illegible]   Data Flow(s) without a description value.                                           Low
 [illegible]   Activitie(s) with timeout values still set to the service default value of 7 days.  High
 [illegible]   Activitie(s) without a description value.                                           Low
 [illegible]   Activitie(s) ForEach iteration without a batch count value set.                     High
 [illegible]   Activitie(s) ForEach iteration with a batch count size that is less than the service maximum. Medium
 [illegible]   Linked Service(s) not using Azure Key Vault to store credentials.                   High
 [illegible]   Linked Service(s) not used by any other resource.                                   Medium
 [illegible]   Linked Service(s) without a description value.                                      Low
 [illegible]   Linked Service(s) without annotations.                                              Low
 [illegible]   Dataset(s) not used by any other resource.                                          Medium
 [illegible]   Dataset(s) without a description value.                                             Low
 [illegible]   Dataset(s) not organised into folders.                                              Low
 [illegible]   Dataset(s) without annotations.                                                     Low
 [illegible]   Trigger(s) not used by any other resource.                                          Medium
 [illegible]   Trigger(s) without a description value.                                             Low
 [illegible]   Trigger(s) without annotations.                                                     Low
```

(https://mrpaulandrew.files.wordpress.com/2020/11/adf-checker-output-v0.1-1.png)

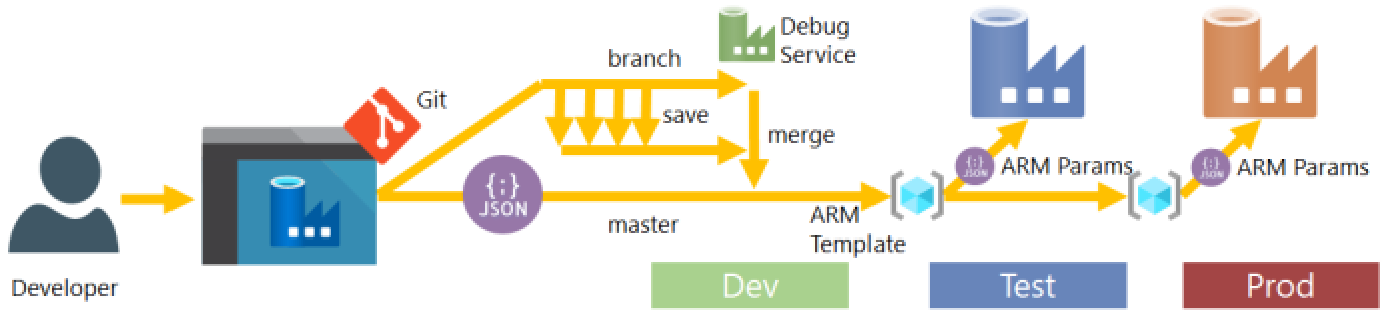
Let's start, my set of Data Factory best practices:

Platform Setup

Environment Setup & Developer Debugging

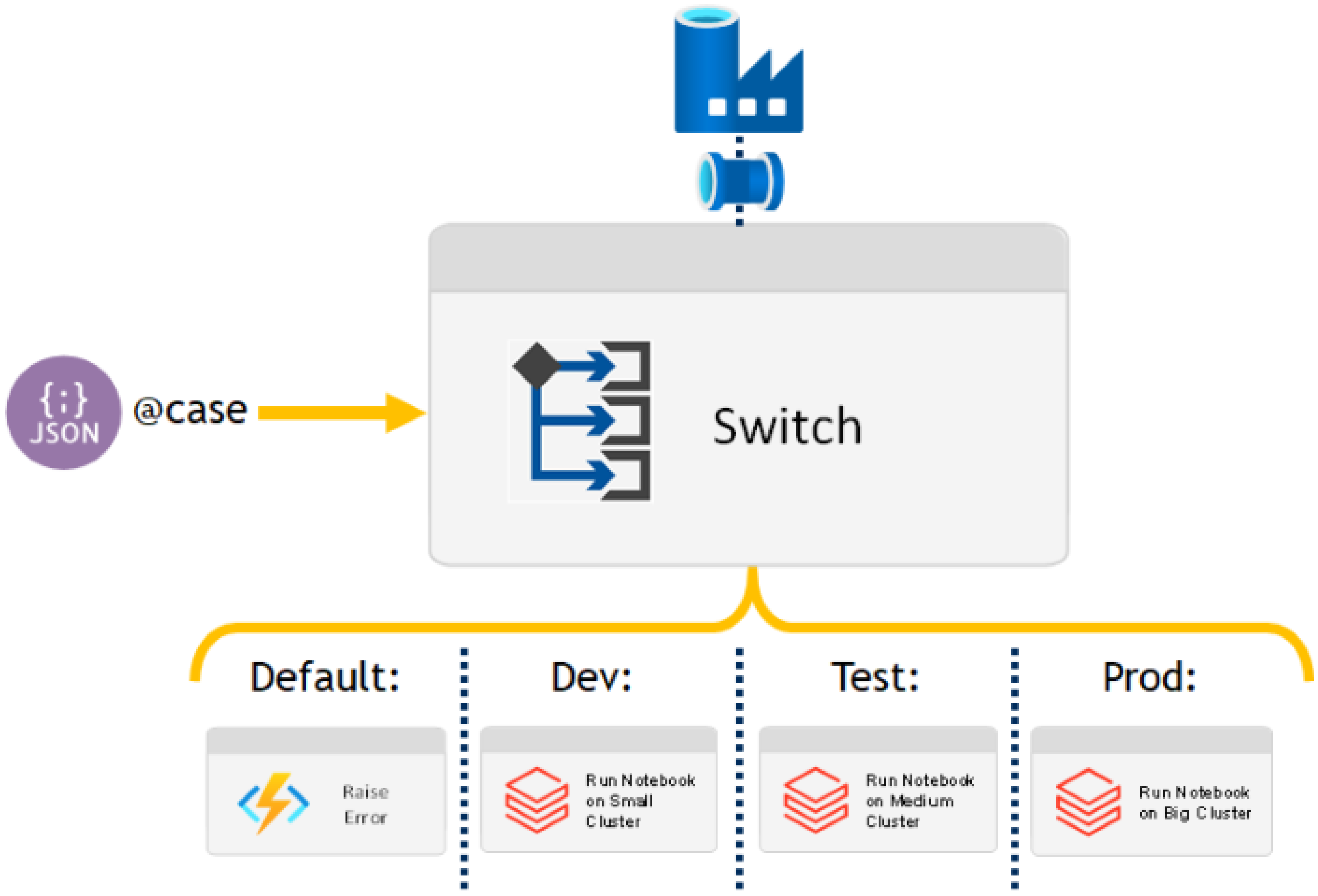
Having a clean separation of resources for development, testing and production. Obvious for any solution, but when applying this to ADF, I'd expect to see the development service connected to source control as a minimum. Using Azure DevOps or GitHub doesn't matter, although authentication against Azure DevOps is slightly simpler within the same tenant. Then, that development service should be used with multiple code repository branches that align to backlog features. Next, I'd expect developers working within those code branches to be using the ADF debug feature to perform basic end to end testing of newly created pipelines and using break points on activities as required. Pull requests of feature branches would be peer reviewed before merging into the main delivery branch and published to the development Data Factory service. Having that separation of debug and development is important to understand for that first Data Factory service and even more important to get it connected to a source code system.

For clarification, other downstream environments (test, UAT, production) do not need to be connected to source control.



(https://mrpaulandrew.files.wordpress.com/2019/12/adf-arm-deployments.png)

Final thoughts on environment setup. Another option and technique I've used in the past is to handle different environment setups internally to Data Factory via a Switch activity. In this situation a central variable controls which activity path is used at runtime. For example, having different Databricks clusters and Linked Services connected to different environment activities:


{:} JSON
@case
Switch
Default:
Dev:
Test:
Prod:
Raise Error
Run Notebook on Small Cluster
Run Notebook on Medium Cluster
Run Notebook on Big Cluster

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-swtich-activity.png)

This is probably a special case and nesting activities via a 'Switch' does come with some drawbacks. This is not a best practice, but an alternative approach you might want to consider. I blogged about this in more detail here (https://mrpaulandrew.com/2020/01/22/using-the-azure-data-factory-switch-activity/).

Multiple Data Factory Instance's

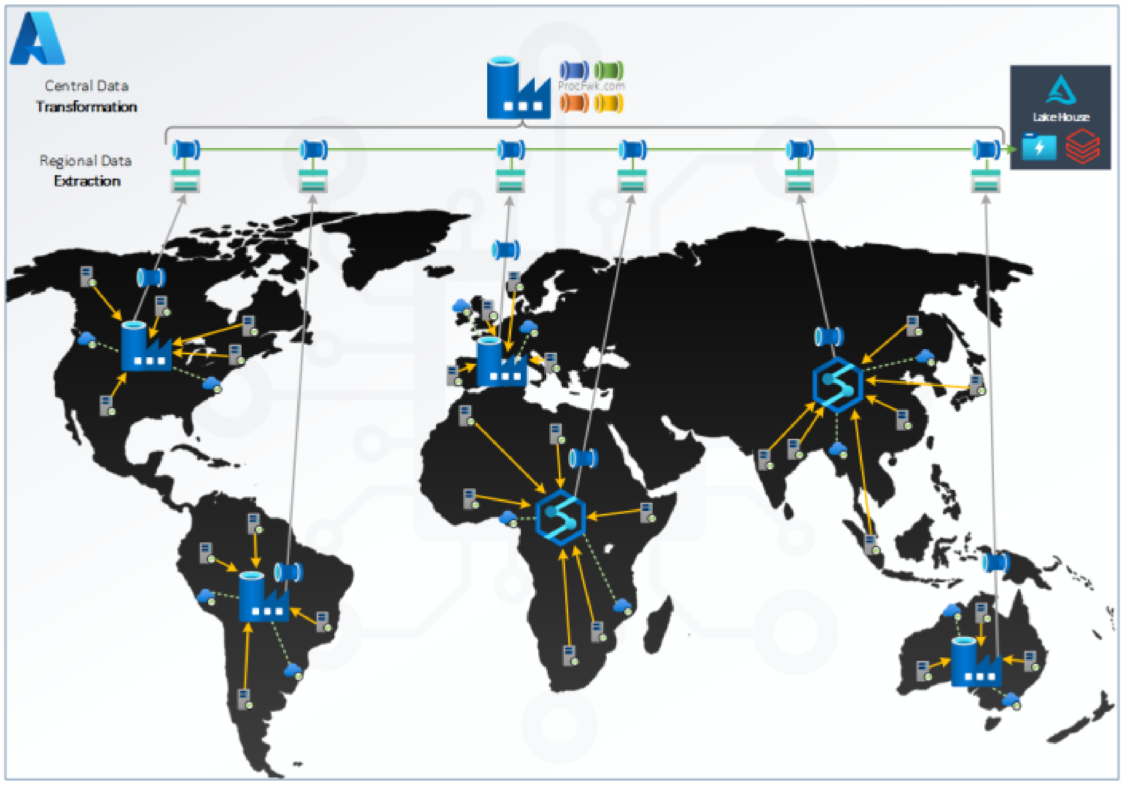
Question: when should I use multiple Data Factory instances for a given solution?

My initial answer(s):

1. To separate business processes (sales, finance, HR).
2. For Azure cost handling and consumption.
3. Due to regional regulations.
4. When handling boiler plate code.
5. For a cleaner security model.
6. Decoupling worker pipelines from wider bootstrap processes.

In all cases these answers aren't hard rules, more a set of guide lines to consider. I've elaborated on each situation as I've encountered them it in the following blog here (https://mrpaulandrew.com/2020/05/22/when-should-i-use-multiple-azure-data-factorys/). Even if you do create multiple Data Factory instances, some resource limitations are handled at the subscription level, so be careful.

Furthermore, depending on the scale of your solution you may wish to check out my latest post on Scaling Data Integration Pipelines here (https://mrpaulandrew.com/2021/08/10/scaling-azure-data-integration-pipelines-decoupling-data-extract-and-transform/). This assumes a large multi national enterprise with data sources deployed globally that need to be ingested.

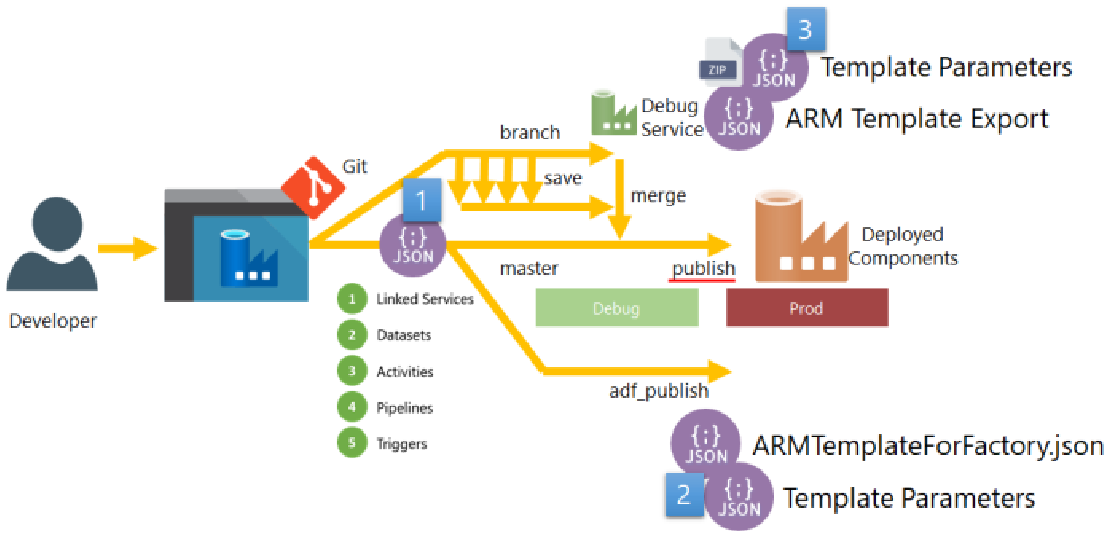


Source code artifacts

With a single Data Factory instance connected to a source code repository its possible to get confused with all the different JSON artifacts available. There are in three possible options here:

1. From the collaboration branch and feature branchs artifacts for each part of the Data Factory instance, separated by sub folders within Git. For example, 1 JSON file per pipeline.
2. If publishing the Data Factory instance via the UI, the publish branch we contain a set of ARM templates, one for the instance and one for all parts of the Data Factory.
3. Via the UI, you can download/export a Zip file containing a different set of ARM templates for the Data Factory instance.

These three options are visualised below.

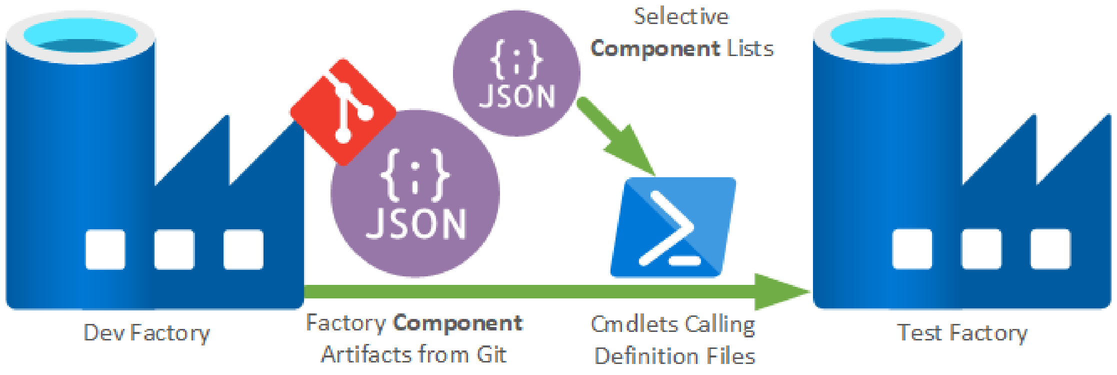


My recommendation is to always use the **option 1 artifacts** as these give you the most granular control over your Data Factory when dealing with pull requests and deployments. Especially for large Data Factory instances with several hundred different pipelines.

Deployments

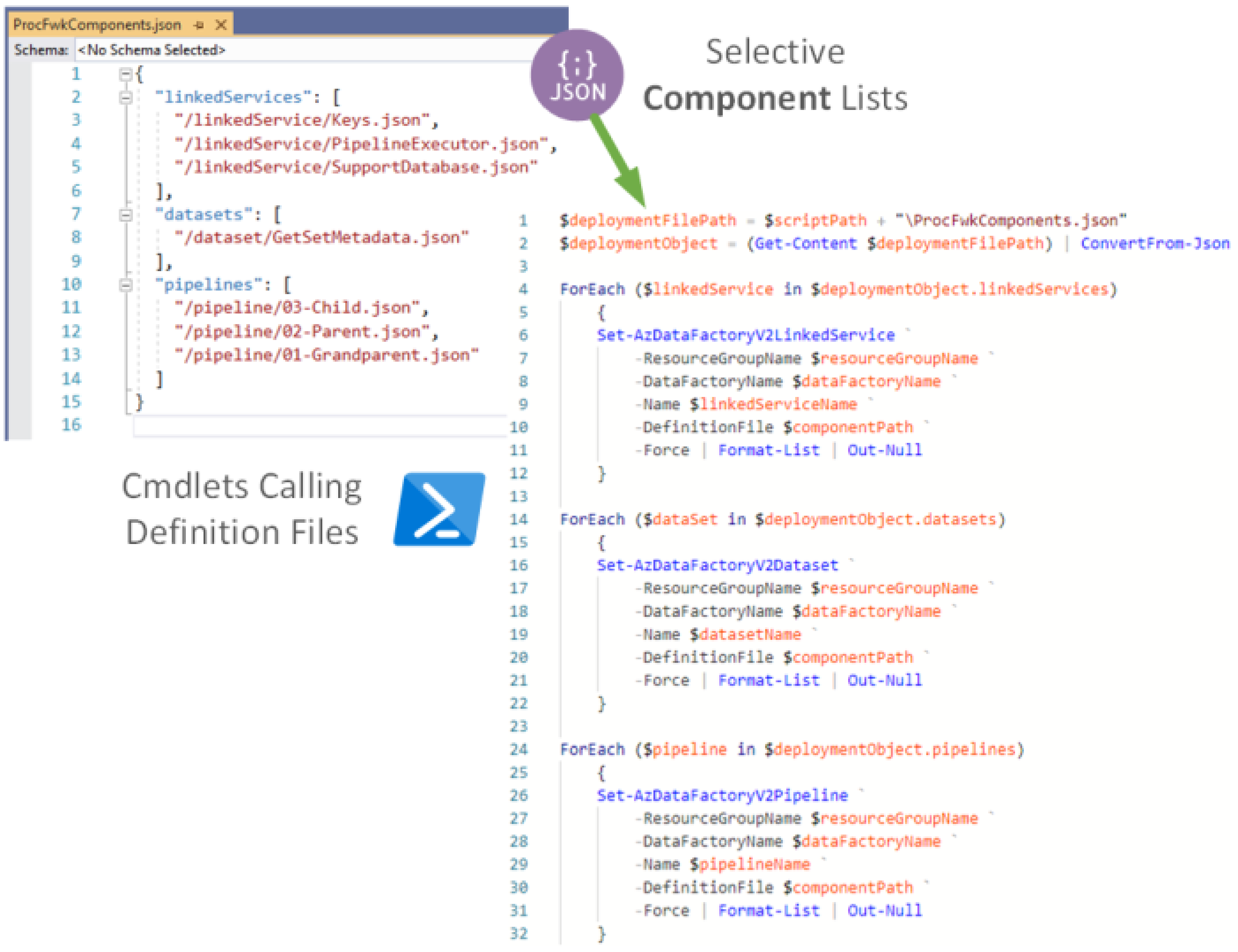
Leading on from our environment setup the next thing to call out is how we handle our Data Factory deployments. The obvious choice might be to use ARM templates. However, this isn't what I'd recommend as an approach (sorry Microsoft). The ARM templates are fine for a complete deployment of everything in your Data Factory, maybe for the first time, but they don't offer any granular control over specific components and by default will only expose Linked Service values as parameters.

My approach for deploying Data Factory would be to use PowerShell cmdlets and the JSON definition files found in your source code repository, this would also be supported by a config file of component lists you want to deploy. Generally this technique of deploying Data Factory parts with a 1:1 between PowerShell cmdlets and JSON files offers much more control and options for dynamically changing any parts of the JSON at deployment time. But, is does mean you have to manually handle component dependencies and removals, if you have any. A quick visual of the approach:



To elaborate, the PowerShell uses the artifacts created by Data Factory in the 'normal' repository code branches (not the **adf_publish** branch), in the section above I refer to this as 'artifacts option 1'. Then for each component provides this via a configurable list as a definition file to the respective PowerShell cmdlets. The cmdlets use the `DefinitionFile` parameter to set exactly what you want in your Data Factory given what was created by the repo connect instance.

Sudo PowerShell and JSON example below building on the visual representation above, click to enlarge.


ProcFwkComponents.json
Schema: <No Schema Selected>
{;} JSON
Selective Component Lists
Cmdlets Calling Definition Files

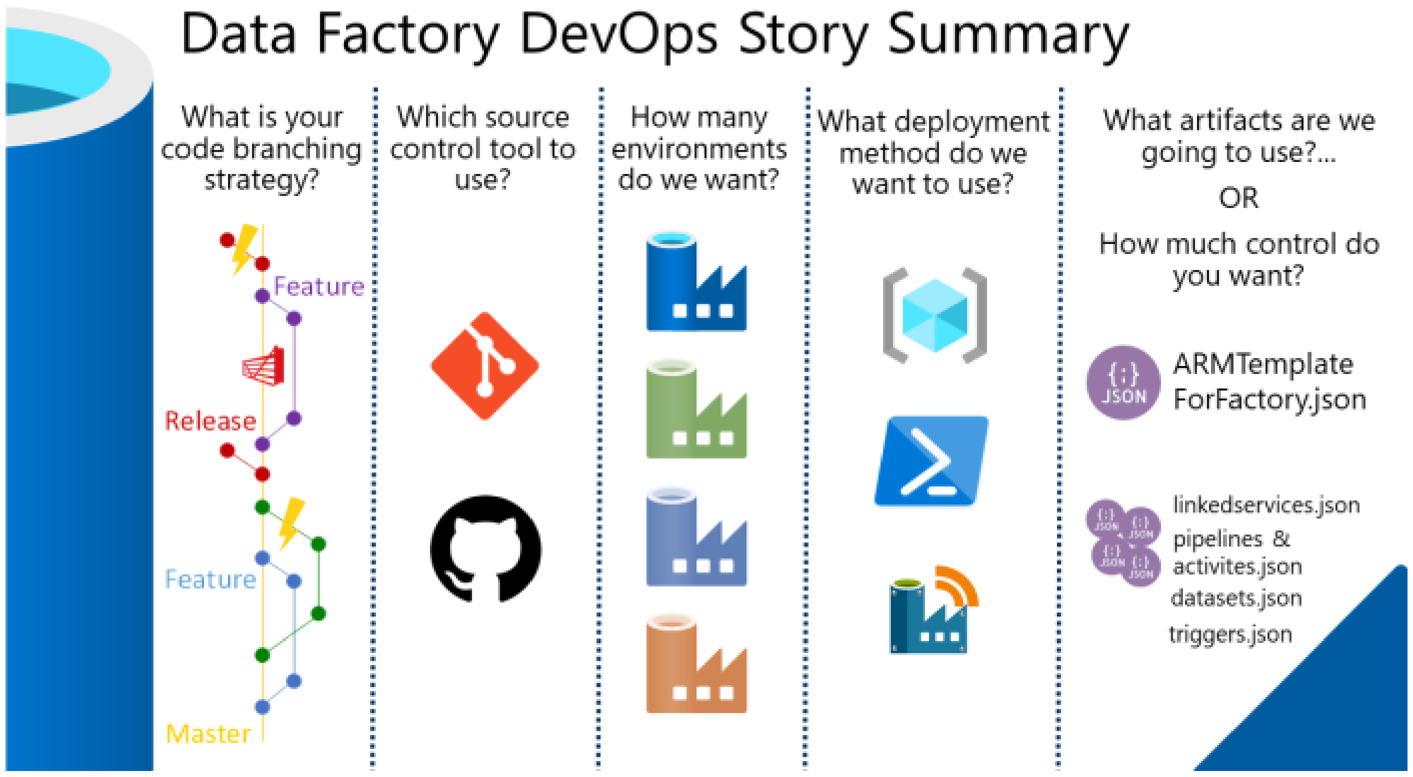
```json
{
  "linkedServices": [
    "/linkedService/Keys.json",
    "/linkedService/PipelineExecutor.json",
    "/linkedService/SupportDatabase.json"
  ],
  "datasets": [
    "/dataset/GetSetMetadata.json"
  ],
  "pipelines": [
    "/pipeline/03-Child.json",
    "/pipeline/02-Parent.json",
    "/pipeline/01-Grandparent.json"
  ]
}
```

```powershell
$deploymentFilePath = $scriptPath + "\ProcFwkComponents.json"
$deploymentObject = (Get-Content $deploymentFilePath) | ConvertFrom-Json

ForEach ($linkedService in $deploymentObject.linkedServices)
    {
    Set-AzDataFactoryV2LinkedService `
        -ResourceGroupName $resourceGroupName `
        -DataFactoryName $dataFactoryName `
        -Name $linkedServiceName `
        -DefinitionFile $componentPath `
        -Force | Format-List | Out-Null
    }

ForEach ($dataSet in $deploymentObject.datasets)
    {
    Set-AzDataFactoryV2Dataset `
        -ResourceGroupName $resourceGroupName `
        -DataFactoryName $dataFactoryName `
        -Name $datasetName `
        -DefinitionFile $componentPath `
        -Force | Format-List | Out-Null
    }

ForEach ($pipeline in $deploymentObject.pipelines)
    {
    Set-AzDataFactoryV2Pipeline `
        -ResourceGroupName $resourceGroupName `
        -DataFactoryName $dataFactoryName `
        -Name $pipelineName `
        -DefinitionFile $componentPath `
        -Force | Format-List | Out-Null
    }
```

(https://mrpaulandrew.files.wordpress.com/2020/04/sudo-code-for-deploying-data-factory-1.png)

If you think you'd like to use this approach, but don't want to write all the PowerShell yourself, great news, my friend and colleague Kamil Nowinski (https://twitter.com/NowinskiK) has done it for you in the form of a PowerShell module (**azure.datafactory.tools**). Check out his GitHub repository here (https://github.com/SQLPlayer/azure.datafactory.tools). This also can now handle dependencies.

I've attempted to summarise the key questions you probably need to ask yourself when thinking about Data Factory deployments in the following slide.

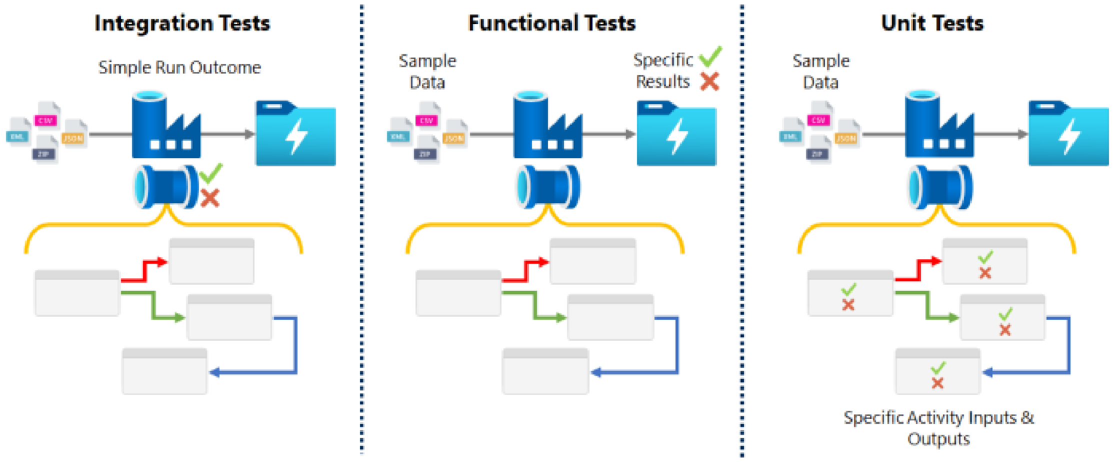

Data Factory DevOps Story Summary
What is your code branching strategy?
Feature
Release
Feature
Master
Which source control tool to use?
How many environments do we want?
What deployment method do we want to use?
What artifacts are we going to use?...
OR
How much control do you want?
ARMTemplate ForFactory.json
linkedservices.json
pipelines & activites.json
datasets.json
triggers.json

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-devops-summary.png)

Automated Testing

To complete our best practices for environments and deployments we need to consider testing. Given the nature of Data Factory as a cloud service and an orchestrator what should be tested often sparks a lot of debate. Are we testing the pipeline code itself, or what the pipeline has done in terms of outputs? Is the business logic in the pipeline or wrapped up in an external service that the pipeline is calling?

The other problem is that a pipeline will need to be published/deployed in your Data Factory instance before any external testing tools can execute it as a pipeline run/trigger. This then leads to a chicken/egg situation of wanting to test before publishing/deploying, but not being able to access your Data Factory components in an automated way via the debug area of the resource.



Currently my stance is simple:

- Perform basic testing using the repository connected Data Factory debug area and development environment.
- Deploy all your components to your Data Factory test instance. This could be in your wider test environment or as a dedicated instance of ADF just for testing publish pipelines.
- Run everything end to end (if you can) and see what breaks.
- Inspect activity inputs and outputs where possible and especially where expressions are influencing pipeline behaviour.

Another friend and ex-colleague Richard Swinbank (https://twitter.com/RichardSwinbank) has a great blog series on running these pipeline tests via an NUnit project in Visual Studio. Check it out here (https://richardswinbank.net/tag/adftesting?do=showtag&tag=adftesting).

Wider Platform Orchestration

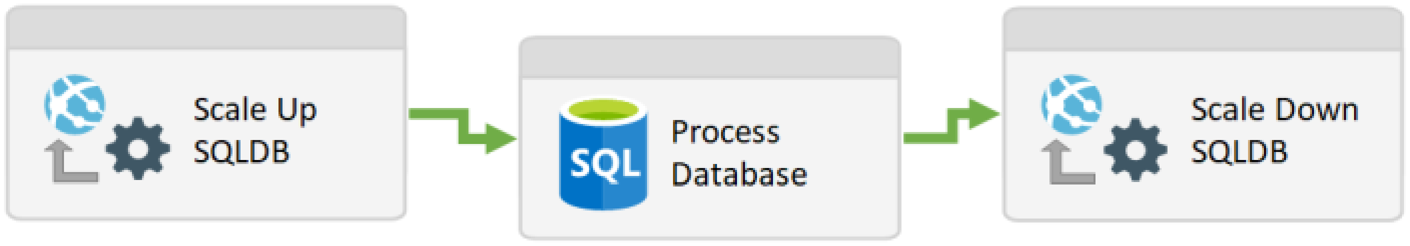
In Azure we need to design for cost, I never pay my own Azure Subscription bills, but even so. We should all feel accountable for wasting money. To that end, pipelines should be created with activities to control the scaling of our wider solution resources.

- For a SQLDB, scale it up before processing and scale it down once finished.
- For a SQLDW (Synapse SQL Pool), start the cluster before processing, maybe scale it out too. Then pause it after.
- For Analysis service, resume the service to process the models and pause it after. Maybe, have a dedicated pipeline that pauses the service outside of office hours.
- For Databricks, create a linked services that uses job clusters.
- For Function Apps, consider using different App Service plans and make best use of the free consumption (compute) offered where possible.

You get the idea. Check with the bill payer, or pretend you'll be getting the monthly invoice from Microsoft.

Building pipelines that don't waste money in Azure Consumption costs is a practice that I want to make the technical standard, not best practice, just normal and expected in a world of 'Pay-as-you-go' compute.

I go into greater detail on the SQLDB example in a previous blog post, here (https://mrpaulandrew.com/2019/06/18/azure-data-factory-web-hook-vs-web-activity/).



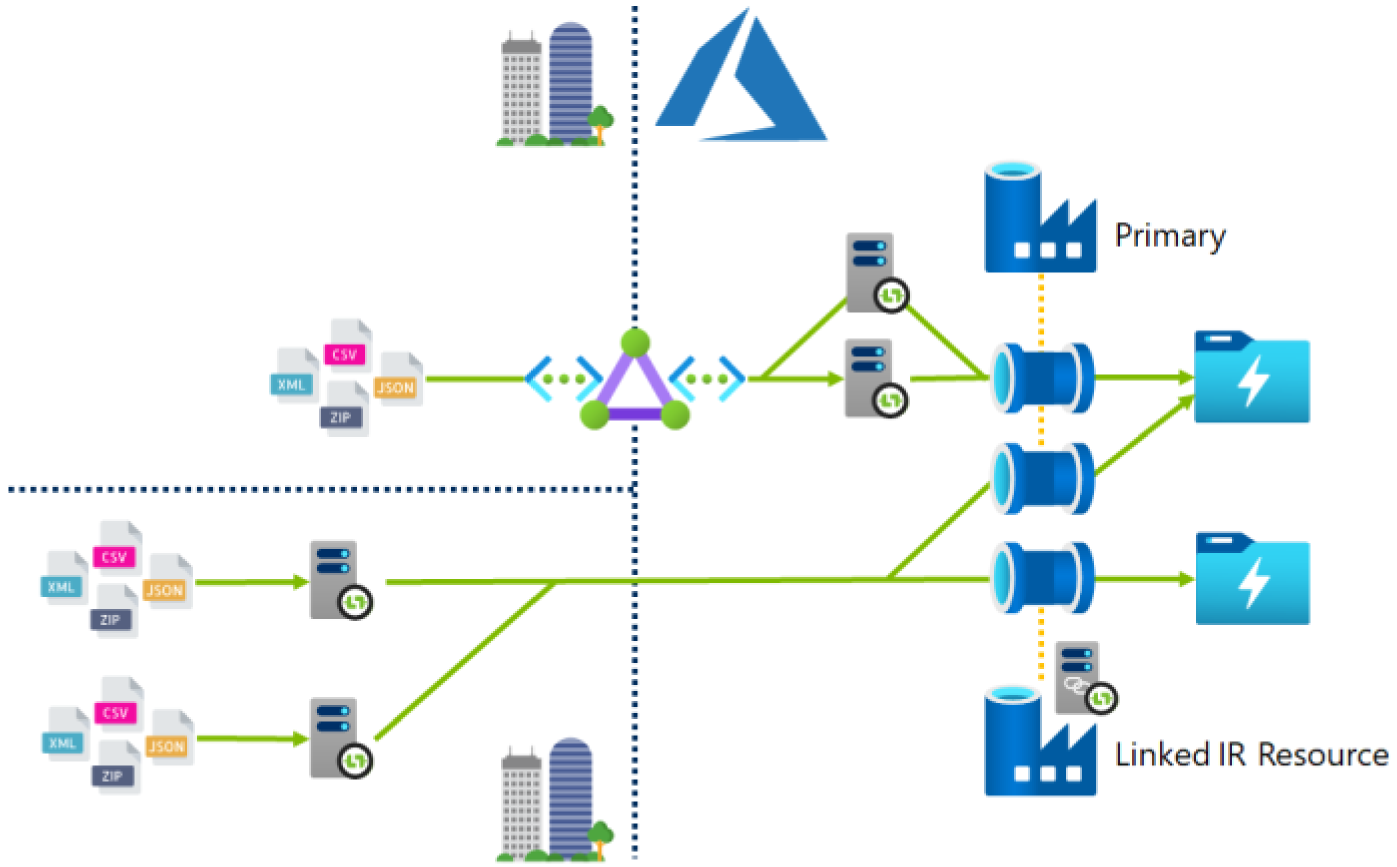
(https://mrpaulandrew.files.wordpress.com/2019/12/scaling-sqldb.png)

Hosted Integration Runtimes

Currently if we want Data Factory to access our on premises resources we need to use the Hosted Integration runtime (previously called the Data Management Gateway in v1 of the service). When doing so I suggest the following two things be taken into account as good practice:

1. Add multiple nodes to the hosted IR connection to offer the automatic failover and load balancing of uploads. Also, make sure you throttle the currency limits of your secondary nodes if the VM's don't have the same resources as the primary node. More details here: https://docs.microsoft.com/en-us/azure/data-factory/create-self-hosted-integration-runtime (https://docs.microsoft.com/en-us/azure/data-factory/create-self-hosted-integration-runtime)
2. When using Express Route or other private connections make sure the VM's running the IR service are on the correct side of the network boundary. If you upgrade to Express Route later in the project and the Hosted IR's have been installed on local Windows boxes, they will probably need to be moved. Consider this in your future architecture and upgrade plans.

For larger deployments and Azure estates consider the wider landscape with multiple IR's being used in a variety of ways. Data Factory might be a PaaS technology, but handling Hosted IRs requires some IaaS thinking and management.



(https://mrpaulandrew.files.wordpress.com/2019/12/adf-uploading-data-via-hosted-irs.png)

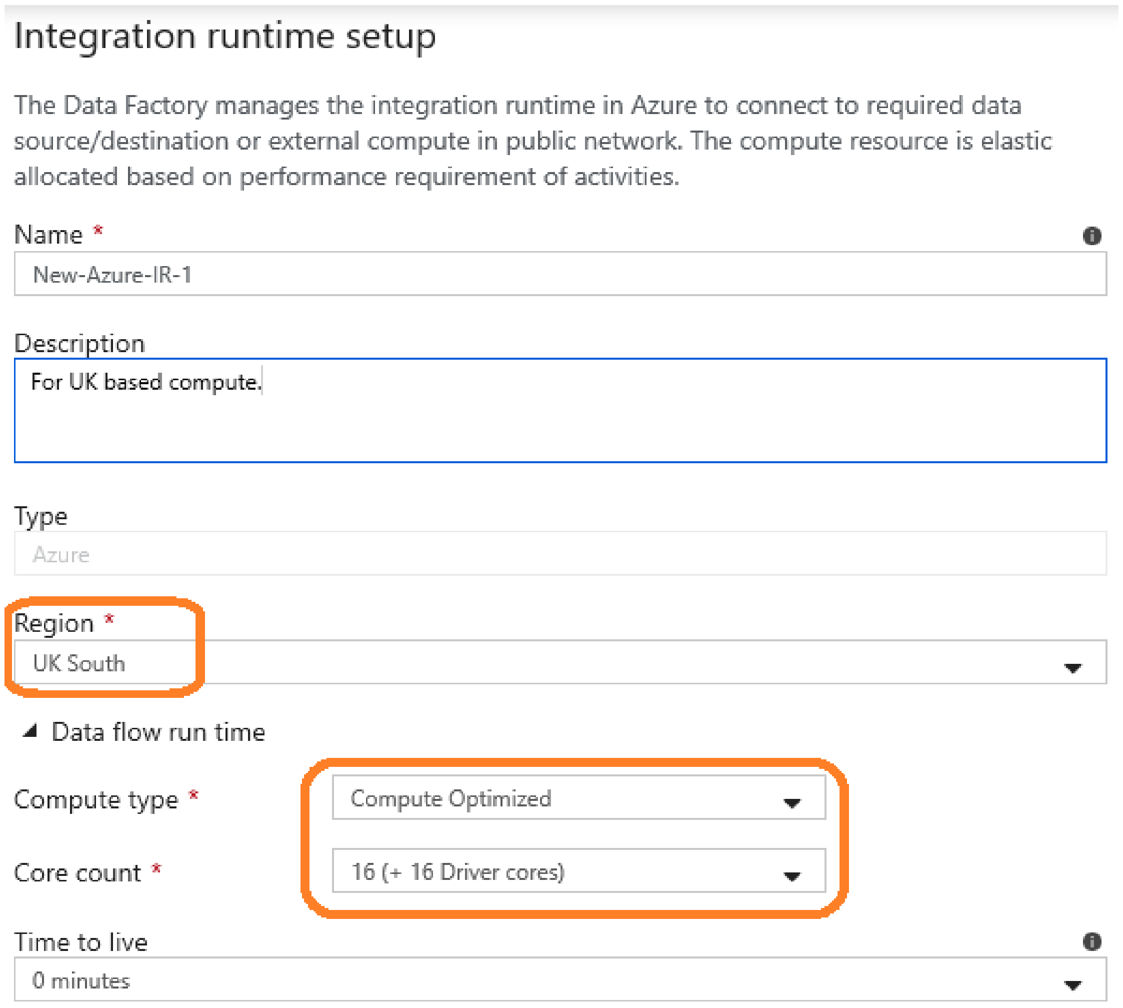
Lastly, make sure in your non functional requirements you capture protentional IR job concurrency. If all job slots are full queuing Activities will start appearing in your pipelines really start to slow things down.

Azure Integration Runtimes

When turning our attention to the Azure flavour of the Integration Runtime I typically like to update this by removing its freedom to auto resolve to any Azure Region. There can be many reasons for this; regulatory etc. But as a starting point, I simply don't trust it not to charge me data egress costs if I know which region the data is being stored.

Also, given the new Data Flow features of Data Factory we need to consider updating the cluster sizes set and maybe having multiple Azure IR's for different Data Flow workloads.

In both cases these options can easily be changed via the portal and a nice description added. Please make sure you tweak these things before deploying to production and align Data Flows to the correct clusters in the pipeline activities.



Integration runtime setup

The Data Factory manages the integration runtime in Azure to connect to required data source/destination or external compute in public network. The compute resource is elastic allocated based on performance requirement of activities.

Name *: New-Azure-IR-1

Description: For UK based compute.

Type: Azure

Region *: UK South

Data flow run time

Compute type *: Compute Optimized

Core count *: 16 (+ 16 Driver cores)

Time to live: 0 minutes

(https://mrpaulandrew.files.wordpress.com/2019/12/manually-set-adf-ir.png)

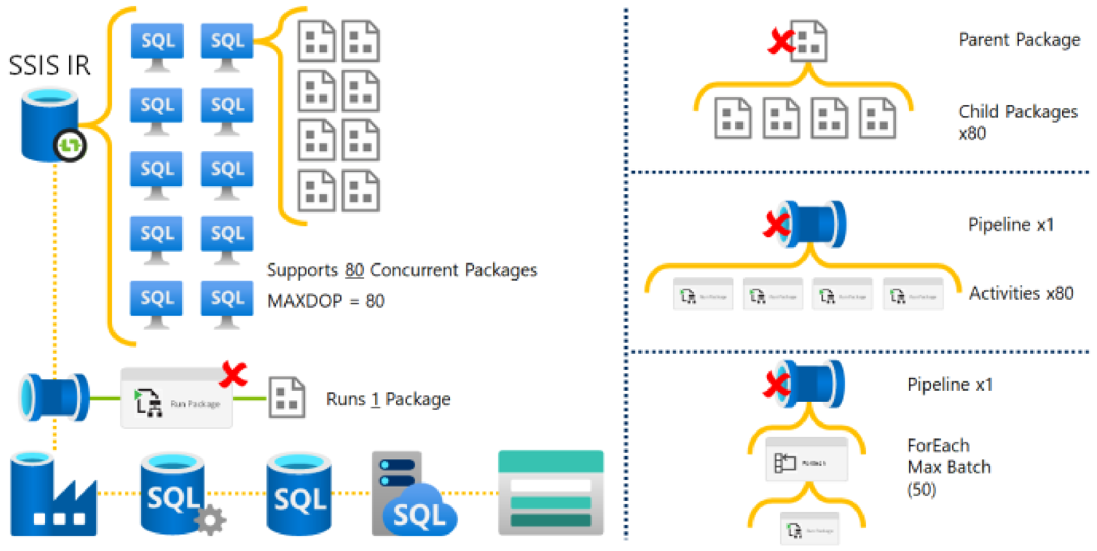
Finally, be aware that the IR's need to be set at the linked service level. Although we can more naturally think of them as being the compute used in our Copy activity, for example.

SSIS Integration Runtimes

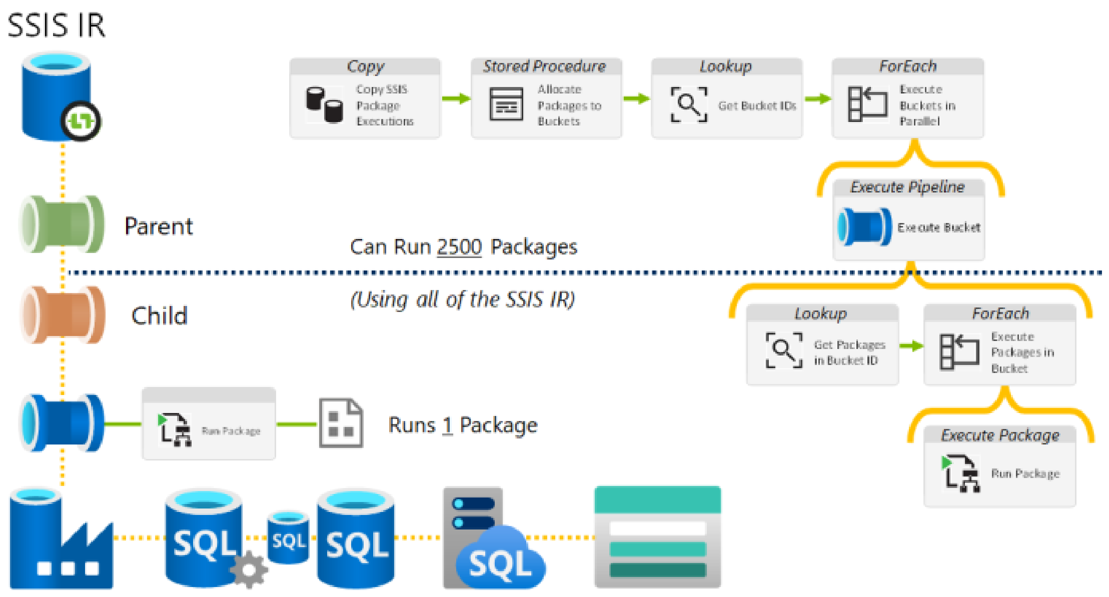
If you have a legacy on premises SQL Server solution that needs migrating into Azure using the PaaS wrapping within Data Factory can be a good hybrid stepping-stone for the project as a middle phase on the way to a full cloud native solution. Specifically thinking about the data transformation work still done by a given SSIS package.

In this context, be mindful of scaling out the SSIS packages on the Data Factory SSIS IR. The IR can support 10x nodes with 8x packages running per node. So a MAXDOP for packages of 80.

That said, there isn't a natural way in Data Factory to run 80 SSIS package activities in parallel, meaning you will be waste a percentage of your SSIS IR compute.



The best solution to this is using nested levels of ForEach activities, combined with some metadata about the packages to scale out enough, that all of the SSIS IR compute is used at runtime.

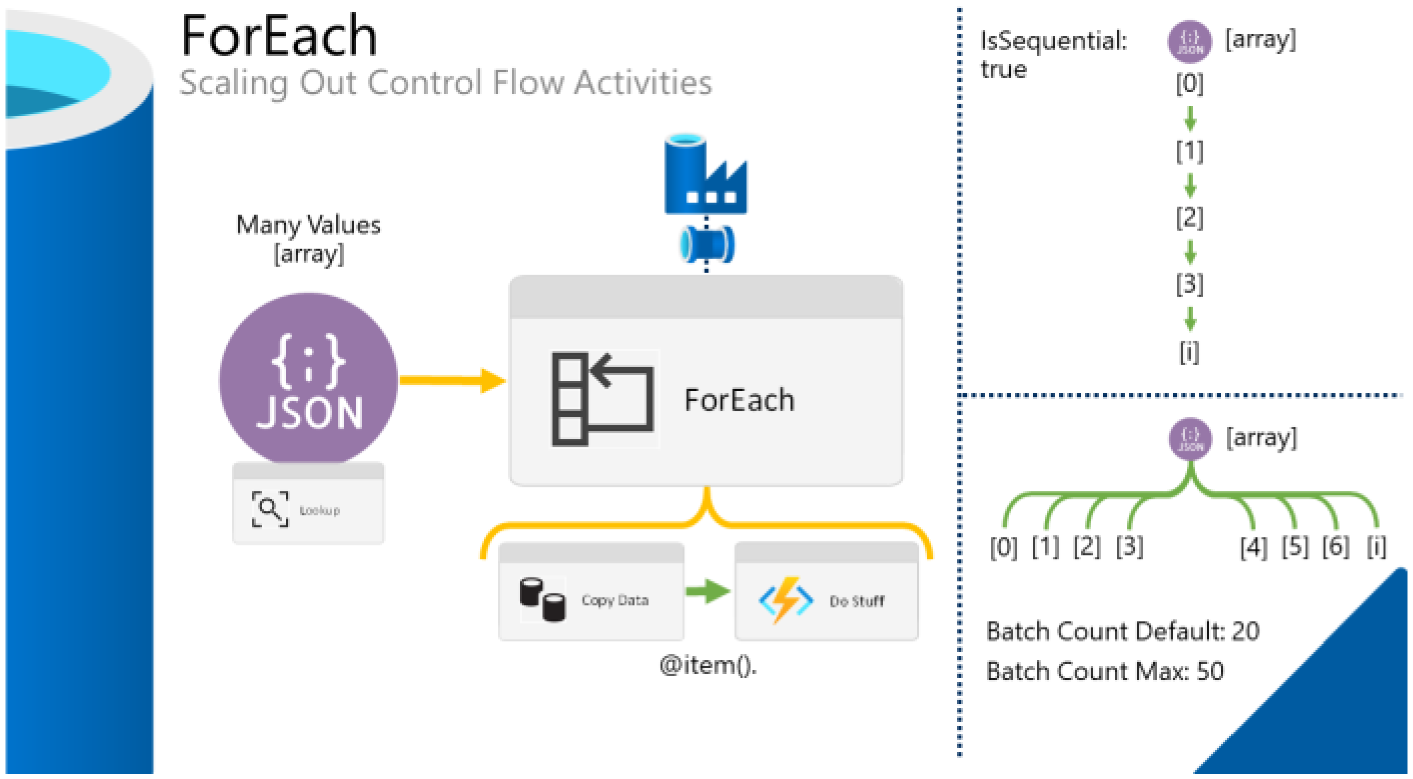


Parallelism

Parallel Execution

Given the scalability of the Azure platform we should utilise that capability wherever possible. When working with Data Factory the 'ForEach' activity is a really simple way to achieve the parallel execution of its inner operations. By default, the ForEach activity does not run sequentially, it will spawn 20 parallel threads and start them all at once. Great! It also has a maximum batch count of 50 threads if you want to scale things out even further. I recommend taking advantage of this behaviour and wrapping all pipelines in ForEach activities where possible.

https://docs.microsoft.com/en-us/azure/data-factory/control-flow-for-each-activity (https://docs.microsoft.com/en-us/azure/data-factory/control-flow-for-each-activity)



(https://mrpaulandrew.files.wordpress.com/2019/12/adf-foreach-slide.png)

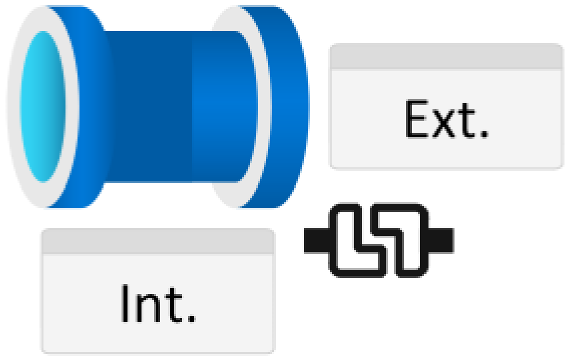
In case you weren't aware within the ForEach activity you need to use the syntax **@{item().SomeArrayValue}** to access the iteration values from the array passed as the ForEach input.

Service Limitations

Please be aware that Azure Data Factory does have limitations. Both internally to the resource and across a given Azure Subscription. When implementing any solution and set of environments using Data Factory please be aware of these limits. To raise this awareness I created a separate blog post about it here (https://mrpaulandrew.com/2020/01/29/azure-data-factory-resource-limitations/) including the latest list of conditions.

The limit I often encounter is where you can only have 40 activities per pipeline. Of course, with metadata driven things this is easy to overcome or you could refactor pipelines in parent and children as already mentioned above. As a best practice, just be aware and be careful. Maybe also check with Microsoft what are hard limits and what can easily be adjusted via a support ticket.

Understanding Activity Concurrency (Internal & External Activities)



As part of Data Factory's service limitations called out in the above section activity concurrency is featured. However, understanding the implications of this can be tricky as the limitation applies **per subscription per IR region**. Not per Data Factory.

To recap:

- For external activities, the limitation is 3,000.
- For internal activities, the limitation is 1,000.

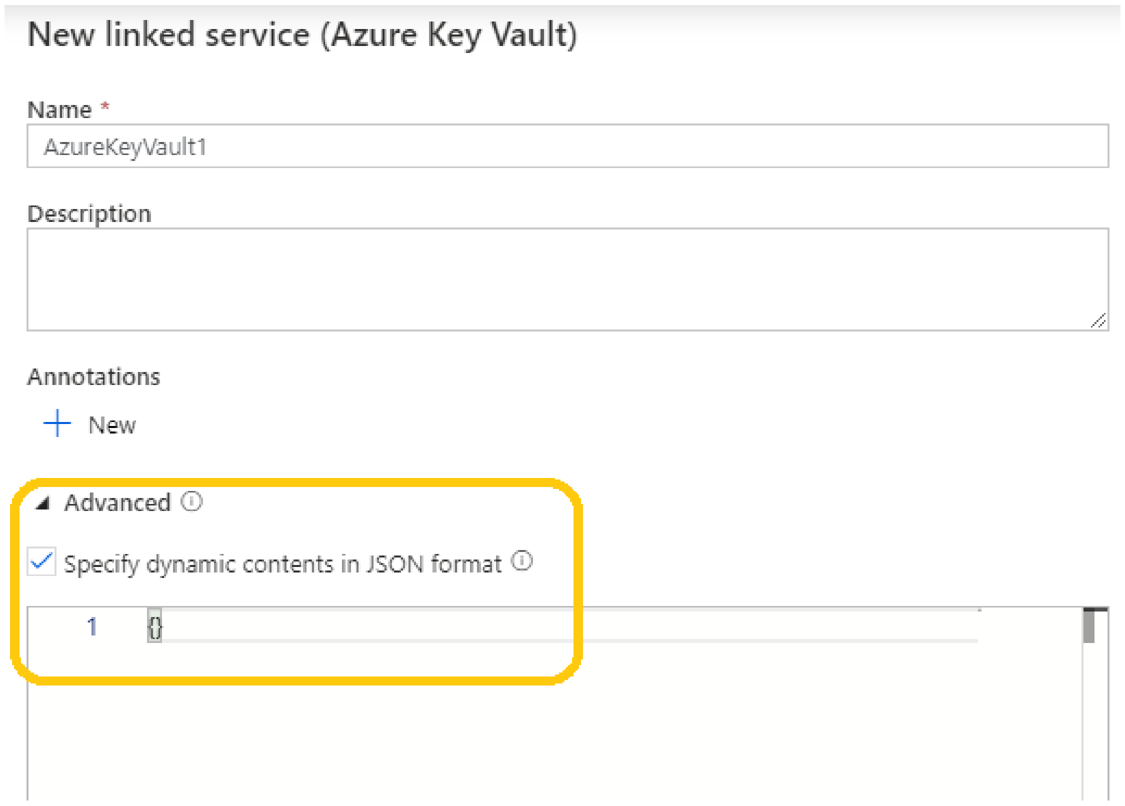
I blogged about this here (https://mrpaulandrew.com/2020/12/22/pipelines-understanding-internal-vs-external-activities/) with a full list of all activities detailing which is internal and which is external.

For a given Data Factory instance you can have multiple IR's fixed to different Azure Regions, or even better, Self Hosted IR's for external handling, so with a little tunning these limits can be overcome. With the caveat that you have good control over all pipeline parallel executions including there inner activity types.

Reusable Code

Dynamic Linked Services

Reusing code is always a great time savers and means you often have a smaller foot print to update when changes are needing. With Data Factory linked services add dynamic content was only supported for a handful of popular connection types. However, now we can make all linked services dynamic (as required) using the feature and tick box called '**Specify dynamic contents in JSON format**'. I've blogged about using this option in a separate post here (https://mrpaulandrew.com/2020/07/14/how-to-use-specify-dynamic-contents-in-json-format-in-azure-data-factory-linked-services/).


New linked service (Azure Key Vault)

Name *
AzureKeyVault1

Description

Annotations
+ New

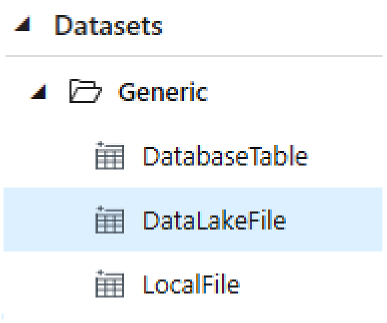
Advanced
✓ Specify dynamic contents in JSON format

```
{}
```

(https://mrpaulandrew.files.wordpress.com/2020/07/adf-ls-dynamic-content-kv-tick-1.png)

Create your complete linked service definitions using this option and expose more parameters in your pipelines to complete the story for dynamic pipelines.

Generic Datasets



(https://mrpaulandrew.files.wordpress.com/2019/12/genericdataset.png)Where design allows it I always try to simplify the number of datasets listed in a Data Factory. In version 1 of the resource separate hard coded datasets were required as the input and output for every stage in our processing pipelines. Thankfully those days are in the past. Now we can use a completely metadata driven dataset for dealing with a particular type of object against a linked service. For example, one dataset of all CSV files from Blob Storage and one dataset for all SQLDB tables.

At runtime the dynamic content underneath the datasets are created in full so monitoring is not impacted by making datasets generic. If anything, debugging becomes easier because of the common/reusable code.

Where generic datasets are used I'd expect the following values to be passed as parameters. Typically from the pipeline, or resolved at runtime within the pipeline.

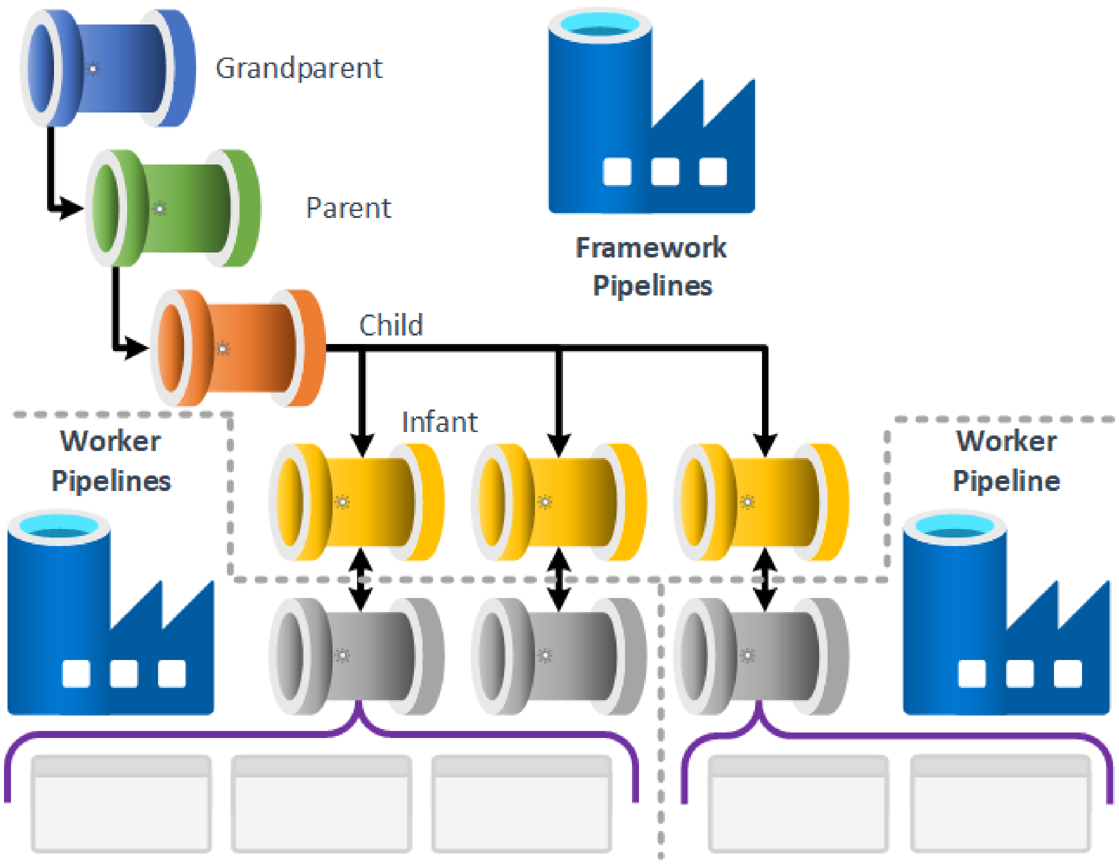
- **Location** – the file path, table location or storage container.
- **Name** – the file or table name.

- **Structure** – the attributes available provided as an array at runtime.

To be clear, I wouldn't go as far as making the linked services dynamic. Unless we were really confident in your controls and credential handling. If you do, the linked service parameters will also need to be addressed, firstly at the dataset level, then in the pipeline activity. It really depends how far you want to go with the parameters.

Pipeline Hierarchies

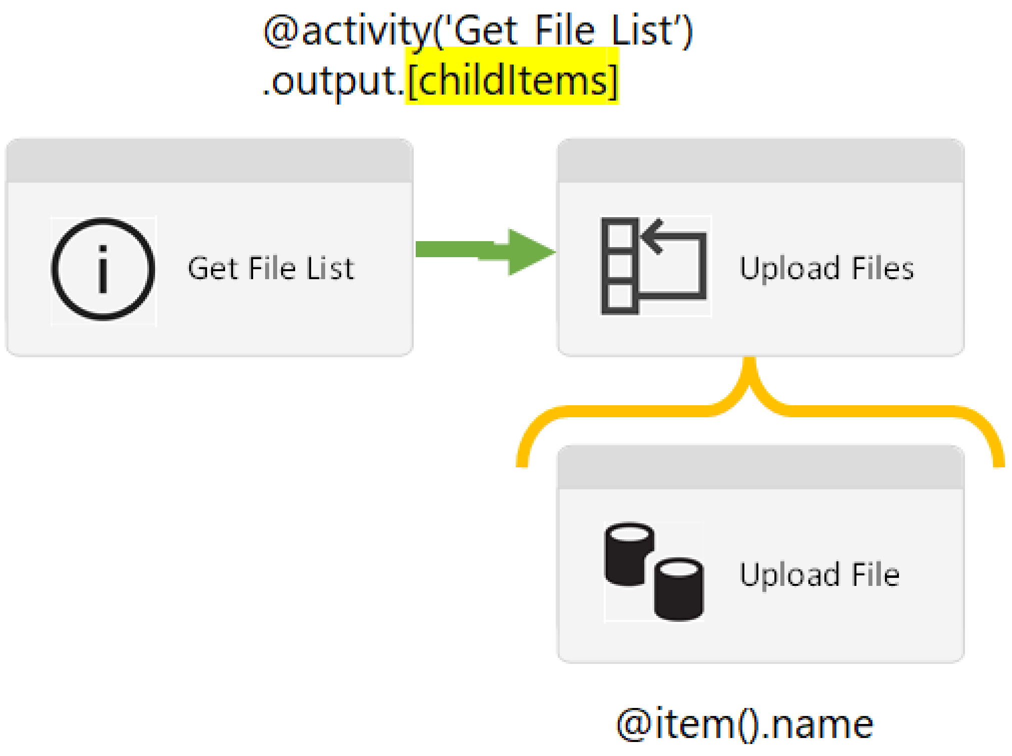
I've blogged about the adoption of pipeline hierarchies as a pattern before (here (https://mrpaulandrew.com/2019/09/25/azure-data-factory-pipeline-hierarchies-generation-control/)) so I won't go into too much detail again. Other than to say its a great technique for structuring any Data Factory, having used it on several different projects. How you define your levels it entirely based on your control flow requirements. Typically though, I'll have at least 4 levels for a large scale solution to control pipeline executions. This doesn't have to be split across Data Factory instances, it depends 🙂



Metadata Driven Processing

Building on our understanding of generic datasets and dynamic linked service, a good Data Factory should include (where possible) generic pipelines, these are driven from metadata to simplify (as a minimum) data ingestion operations. Typically I use an Azure SQLDB to house my metadata with stored procedures that get called via Lookup activities to return everything a pipeline needs to know.

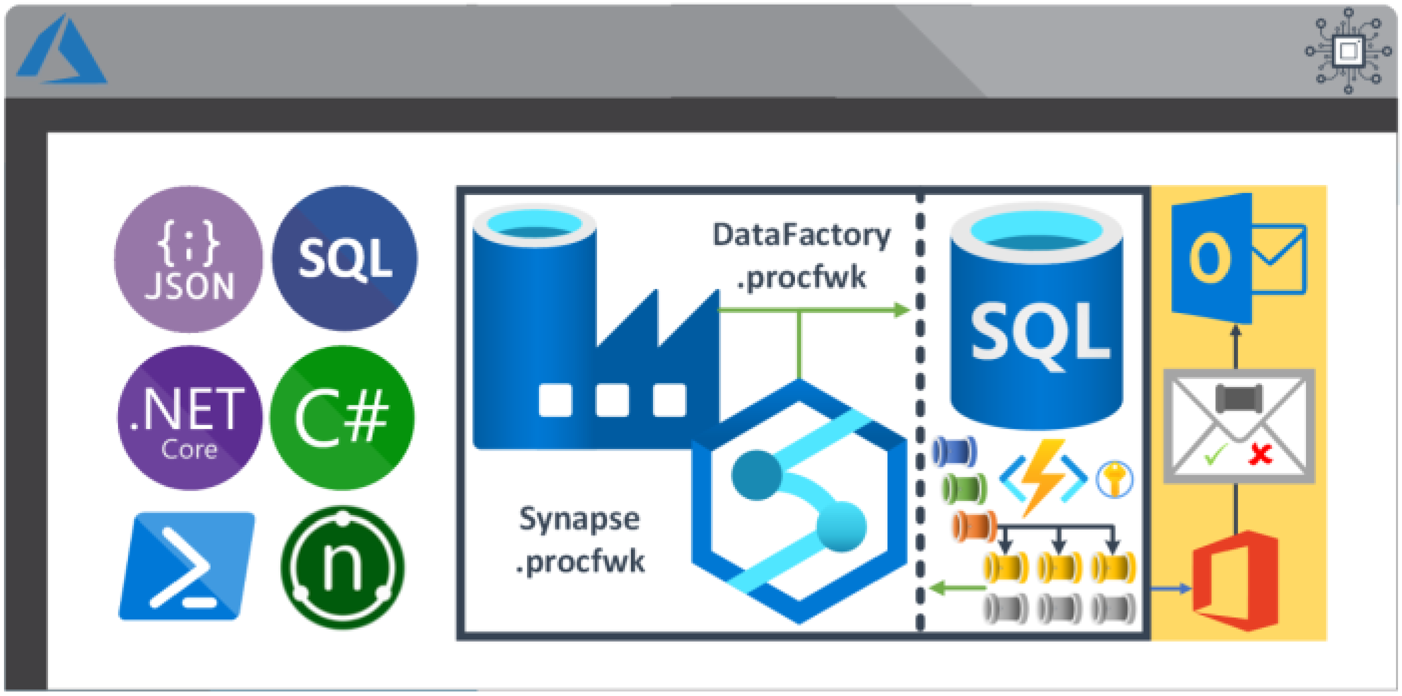
This metadata driven approach means deployments to Data Factory for new data sources are greatly reduced and only adding new values to a database table is required. The pipeline itself doesn't need to be complicated. Copying CSV files from a local file server to Data Lake Storage could be done with just three activities, shown below.



(https://mrpaulandrew.files.wordpress.com/2019/12/adf-metdata-driven-pipeline-example.png)

Building on this I've since created a complete metadata driven processing framework for Data Factory that I call '**procfwk**'. Check out the complete project documentation and GitHub repository if you'd like to adopt this as an open source solution.

- http://procfwk.com (https://mrpaulandrew.github.io/procfwk/)
- https://github.com/mrpaulandrew/procfwk (https://github.com/mrpaulandrew/procfwk)



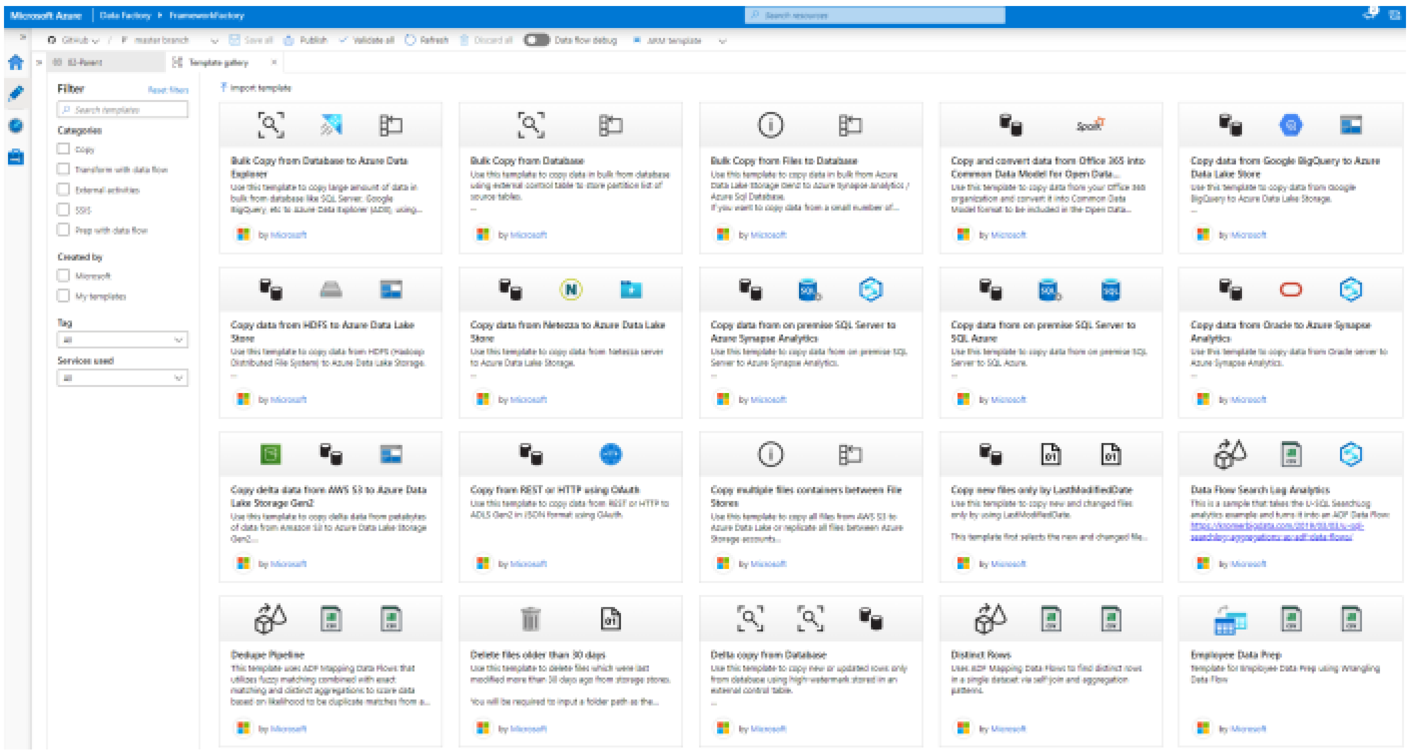
[(https://mrpaulandrew.files.wordpress.com/2019/12/repo-social-media-image.png)](https://mrpaulandrew.files.wordpress.com/2019/12/repo-social-media-image.png)

Using Templates

Pipeline templates I think are a fairly under used feature within Data Factory. They can be really powerful when needing to reuse a set of activities that only have to be provided with new linked service details. And if nothing else, getting Data Factory to create SVG's of your pipelines is really handy for documentation too. I've even used templates in the past to snapshot pipelines when source code versioning wasn't available. A total hack, but it worked well.

Like the other components in Data Factory template files are stored as JSON within our code repository. Each template will have a **manifest.json** file that contains the vector graphic and details about the pipeline that has been captured. Give them a try people.

In addition to creating your own, Data Factory also includes a growing set of common pipeline patterns for rapid development with various activities and data flows.



Bulk Copy from Database to Azure Data Explorer
Use this template to copy large amount of data in bulk from database like SQL Server, Google BigQuery, etc to Azure Data Explorer (ADX), using...
by Microsoft

Bulk Copy from Database
Use this template to copy data in bulk from database using external control table to store partition list of source tables.
...
by Microsoft

Bulk Copy from Files to Database
Use this template to copy data in bulk from Azure Data Lake Storage Gen2 to Azure Synapse Analytics / Azure Sql Database.
If you want to copy data from a small number of...
by Microsoft

Copy and convert data from Office 365 into Common Data Model for Open Data...
Use this template to copy data from your Office 365 organization and convert it into Common Data Model format to be included in the Open Data...
by Microsoft

Copy data from Google BigQuery to Azure Data Lake Store
Use this template to copy data from Google BigQuery to Azure Data Lake Storage.
...
by Microsoft

Copy data from HDFS to Azure Data Lake Store
Use this template to copy data from HDFS (Hadoop Distributed File System) to Azure Data Lake Storage.
...
by Microsoft

Copy data from Netezza to Azure Data Lake Store
Use this template to copy data from Netezza server to Azure Data Lake Storage.
...
by Microsoft

Copy data from on premise SQL Server to Azure Synapse Analytics
Use this template to copy data from on premise SQL Server to Azure Synapse Analytics.
...
by Microsoft

Copy data from on premise SQL Server to SQL Azure
Use this template to copy data from on premise SQL Server to SQL Azure.
...
by Microsoft

Copy data from Oracle to Azure Synapse Analytics
Use this template to copy data from Oracle server to Azure Synapse Analytics.
...
by Microsoft

Copy delta data from AWS S3 to Azure Data Lake Storage Gen2
Use this template to copy delta data from petabytes of data from Amazon S3 to Azure Data Lake Storage Gen2...
by Microsoft

Copy from REST or HTTP using OAuth
Use this template to copy data from REST or HTTP to ADLS Gen2 in JSON format using OAuth.
by Microsoft

Copy multiple files containers between File Stores
Use this template to copy all files from AWS S3 to Azure Data Lake or replicate all files between Azure Storage accounts...
by Microsoft

Copy new files only by LastModifiedDate
Use this template to copy new and changed files only by using LastModifiedDate.

This template first selects the new and changed file...
by Microsoft

Data Flow Search Log Analytics
This is a sample that takes the U-SQL SearchLog analytics example and turns it into an ADF Data Flow: https://kromerbigdata.com/2018/03/03/u-sql-searchlog-aggregations-as-adf-data-flows/
by Microsoft

Dedupe Pipeline
This template uses ADF Mapping Data Flows that utilizes fuzzy matching combined with exact matching and distinct aggregations to score data based on likelihood to be duplicate matches from a...
by Microsoft

Delete files older than 30 days
Use this template to delete files which were last modified more than 30 days ago from storage stores.

You will be required to input a folder path as the...
by Microsoft

Delta copy from Database
Use this template to copy new or updated rows only from database using high-watermark stored in an external control table.
...
by Microsoft

Distinct Rows
Uses ADF Mapping Data Flows to find distinct rows in a single dataset via self-join and aggregation pattern.
by Microsoft

Employee Data Prep
Template for Employee Data Prep using Wrangling Data Flow
by Microsoft

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-template.png)

Security

Linked Service Security via Azure Key Vault

Azure Key Vault is now a core component of any solution, it should be in place holding the credentials for all our service interactions. In the case of Data Factory most Linked Service connections support the querying of values from Key Vault. Where ever possible we should be including this extra layer of security and allowing only Data Factory to retrieve secrets from Key Vault using its own Managed Service Identity (MSI).

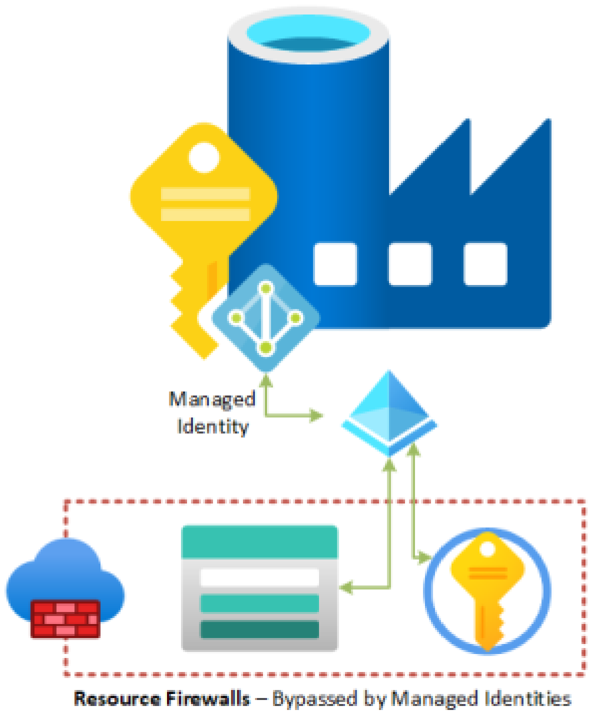
If you aren't familiar with this approach check out this Microsoft Doc pages:

https://docs.microsoft.com/en-us/azure/data-factory/store-credentials-in-key-vault (https://docs.microsoft.com/en-us/azure/data-factory/store-credentials-in-key-vault)

Be aware that when working with custom activities in ADF using Key Vault is essential as the Azure Batch application can't inherit credentials from the Data Factory linked service references.

Managed Identities

A lot of Azure Resources now have their own Managed Identities (MI), previously called Managed Service Identities (MSI's). Microsoft Docs here (https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview).



Using these Managed Identities in the context of Data Factory is a great way to allow interoperability between resources without needing an extra layer of Service Principals (SPN's) or local resource credentials stored in Key Vault.

For example, for Data Factory to interact with an Azure SQLDB, it's Managed Identity can be used as an external identity within the SQL instance. This simplifies authentication massively. Example T-SQL below.

```
CREATE USER [##Data Factory Name (Managed Identity)##]
FROM EXTERNAL PROVIDER;
GO
```

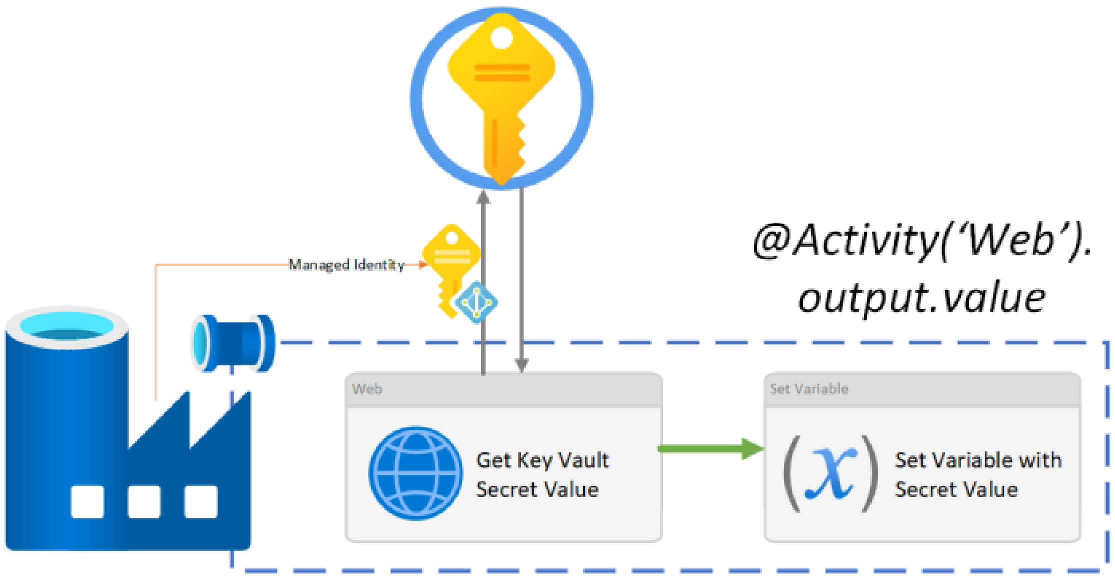
Another good reason for using Managed Identities is that for resources like Storage Accounts, local firewalls can be bypassed if the authentication is done using an MI.

Getting Key Vault secrets at runtime

Using key vault secrets for Linked Service authentication is a given for most connections and a great extra layer of security, but what about within a pipeline execution directly.

Using a Web Activity, hitting the Azure Management API and authenticating via Data Factory's Managed Identity is the easiest way to handle this. See this Microsoft Docs page (https://docs.microsoft.com/en-us/azure/data-factory/how-to-use-azure-key-vault-secrets-pipeline-activities) for exact details.

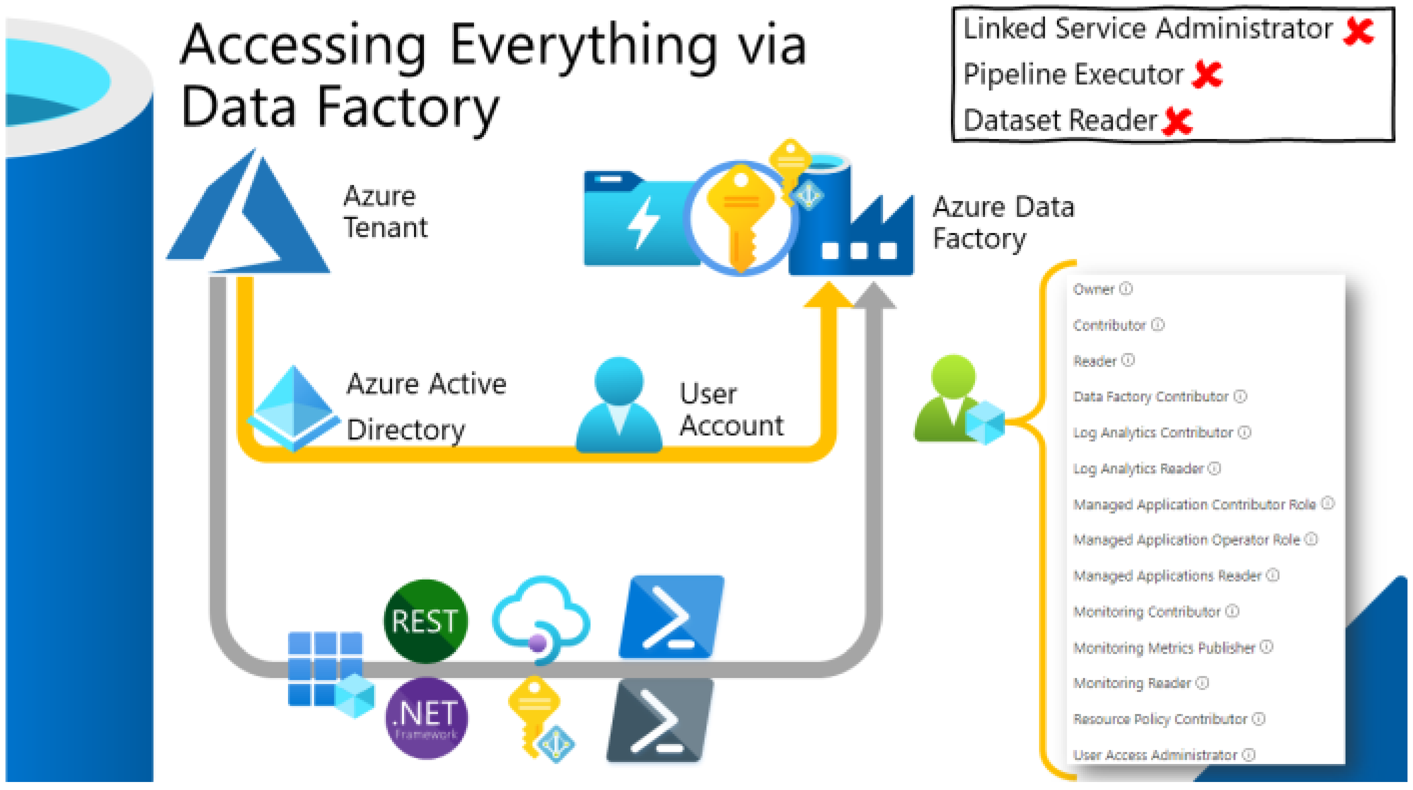
The output of the Web Activity (the secret value) can then be used in all downstream parts of the pipeline.



Security Custom Roles

Given the above stance regarding Azure Key Vault. There is another critical point to consider; Data Factory has access to all the keys in Azure Key Vault. So who/what has access to Data Factory? Are we just moving the point of attack and is Key Vault really adding an extra layer of security?

That said, the security model and roles available for Data Factory are very basic. At the Azure management plane level you can be an Owner or Contributor, that's it. Which in both cases will allow you access to anything in Key Vault using Data Factory as an authentication proxy.


Accessing Everything via Data Factory
Linked Service Administrator
Pipeline Executor
Dataset Reader
Azure Tenant
Azure Data Factory
Azure Active Directory
User Account
REST
.NET
Owner
Contributor
Reader
Data Factory Contributor
Log Analytics Contributor
Log Analytics Reader
Managed Application Contributor Role
Managed Application Operator Role
Managed Applications Reader
Monitoring Contributor
Monitoring Metrics Publisher
Monitoring Reader
Resource Policy Contributor
User Access Administrator

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-security.png)

The be really clear, using Data Factory in debug mode can return a Key Vault secret value really easily using a simple Web Activity request. See Microsoft docs below on doing this:

https://docs.microsoft.com/en-us/azure/data-factory/how-to-use-azure-key-vault-secrets-pipeline-activities (https://docs.microsoft.com/en-us/azure/data-factory/how-to-use-azure-key-vault-secrets-pipeline-activities)

Given this, we should consider adding some custom roles to our Azure Tenant/Subscriptions to better secure access to Data Factory. As a starting point I suggest creating the following custom roles: (https://mrpaulandrew.files.wordpress.com/2019/12/adf-custom-role.png)

- Data Factory Pipeline Executor
- Data Factory Reader

In both cases, users with access to the Data Factory instance can't then get any keys out of Key Vault, only run/read what has already been created in our pipelines. You can find some sample JSON snippets to create these custom roles in my GitHub repository here (https://github.com/mrpaulandrew/BlogSupportingContent/tree/master/Best%20Practices%20for%20Implementing%20Azure%20Data%20Factory/Security%20Custom%20Roles).

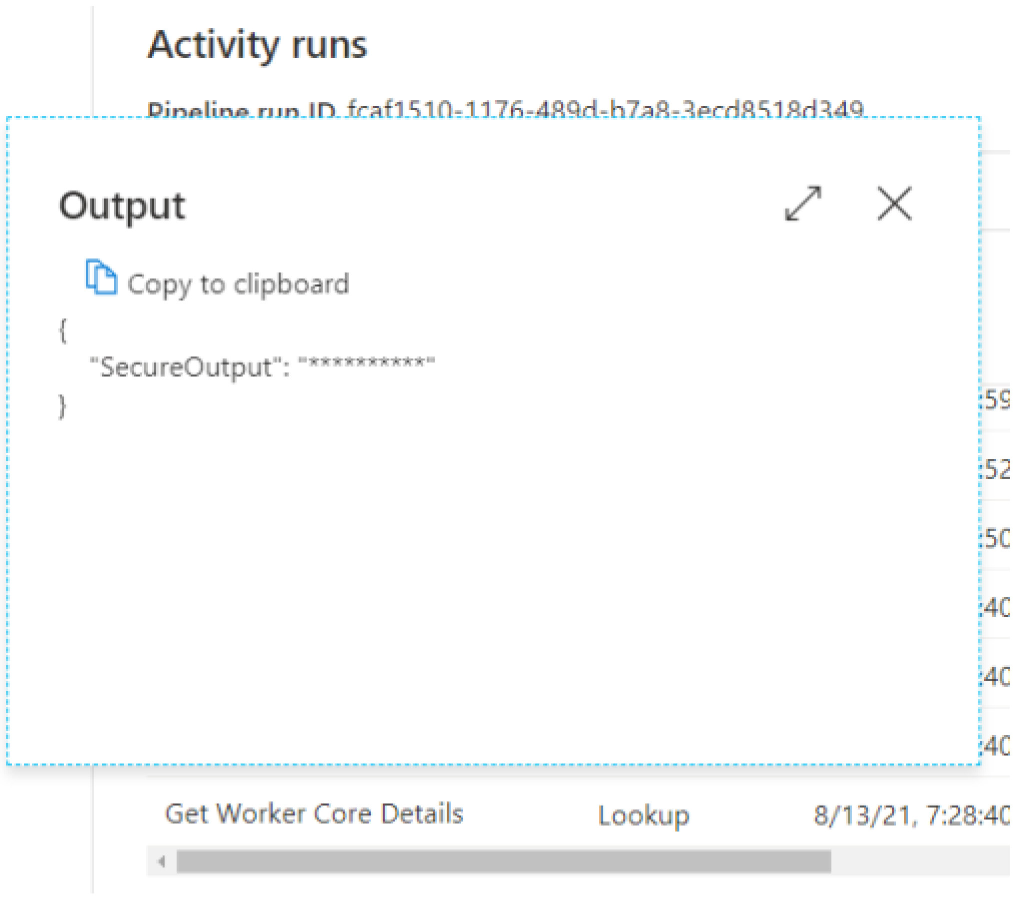
For this, currently you'll require a premium Azure tenant. I also recommend using the Azure CLI to deploy the roles as the PowerShell preview modules and Portal UI screen don't work every well at the point of writing this.

Securing Activity Inputs and Outputs

For the majority of activities within a pipeline having full telemetry data for logging is a good thing. However, for some special cases the output of the activity might become sensitive information that should be visible as plain text.

Maybe a Lookup activity is hitting a SQL Database and returning PII information. Or, even a bear token is being passed downstream in a pipeline for an API call.

In these cases, set the 'Secure Input' and 'Secure Output' attributes for the activity. Shown on the right. These settings are available in Data Factory for most external activities and when looking back in the monitoring have the following affect.


Activity runs

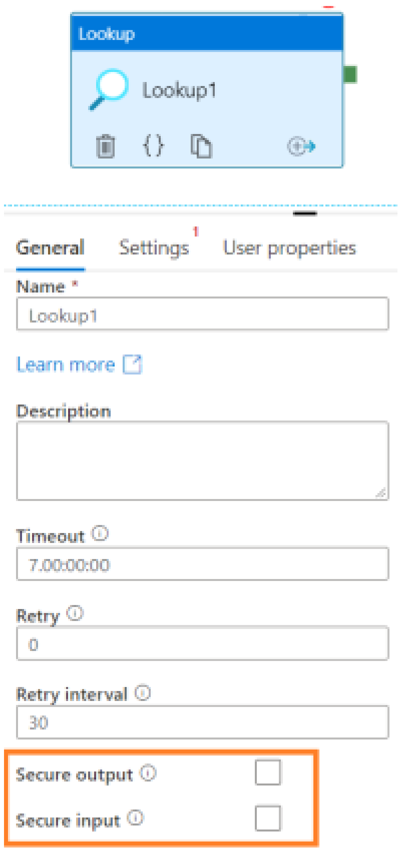
Pipeline run ID fcaf1510-1176-489d-b7a8-3ecd8518d349

Output

Copy to clipboard

```
{
  "SecureOutput": "**********"
}
```

Get Worker Core Details	Lookup	8/13/21, 7:28:40



Name *: Lookup1

Description:

Timeout: 7.00:00:00

Retry: 0

Retry interval: 30

Secure output: ☐

Secure input: ☐

Monitoring & Error Handling

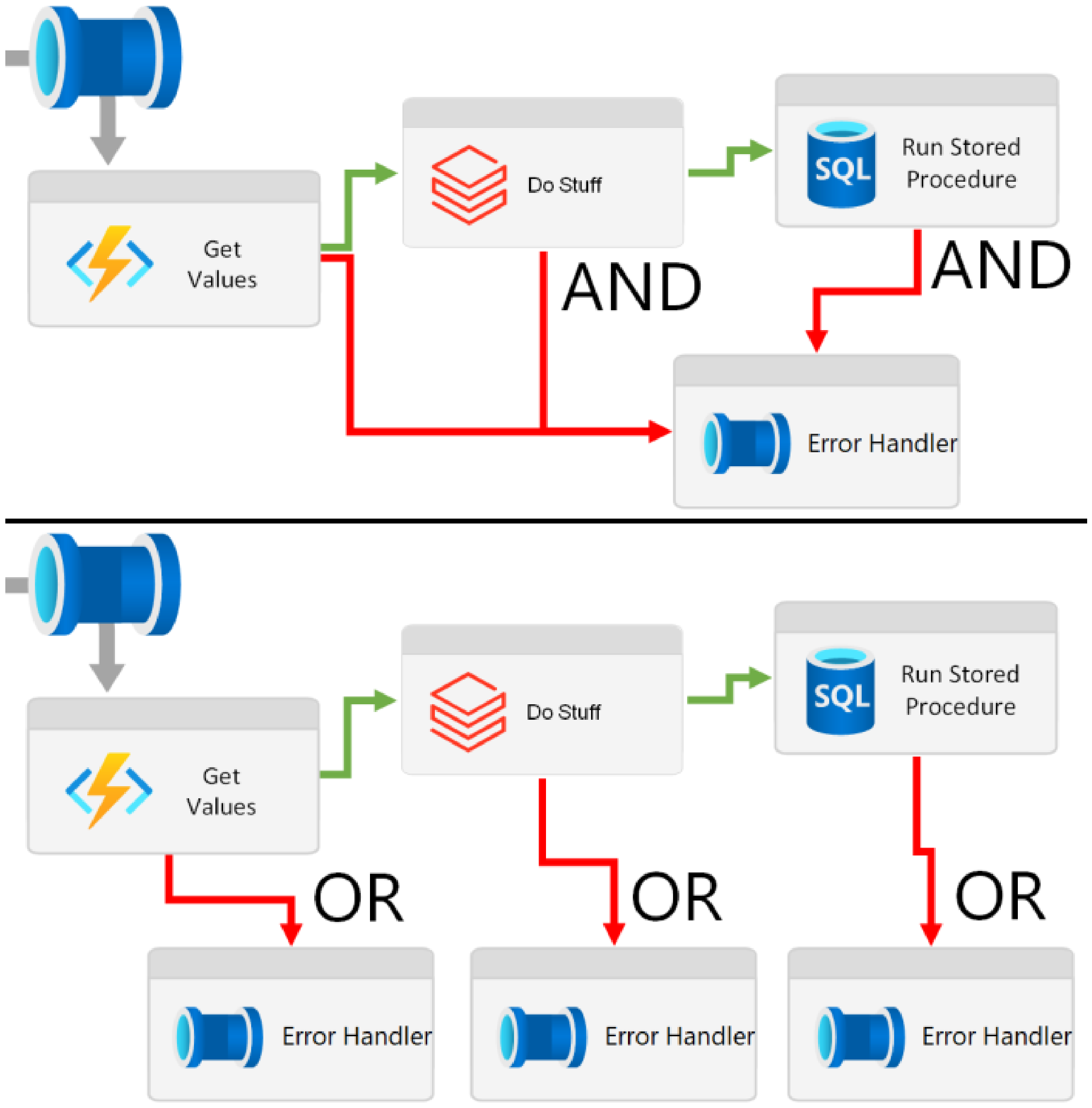
Custom Error Handler Paths

Our Data Factory pipelines, just like our SSIS packages deserve some custom logging and error paths that give operational teams the detail needed to fix failures. For me, these boiler plate handlers should be wrapped up as ‘Infant’ pipelines and accept a simple set of details:

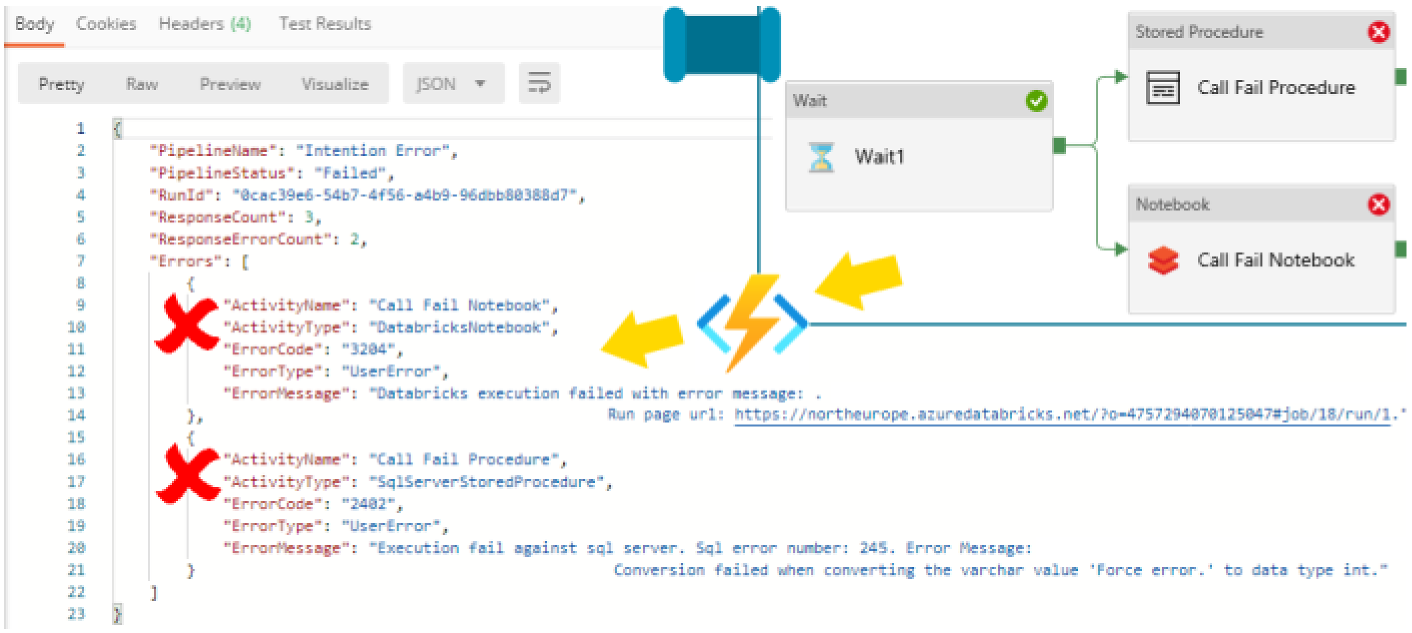
- Calling pipeline name.
- Run ID of the failed execution.
- Custom details coded into the process.

Everything else can be inferred or resolved by the error handler.

Once established, we need to ensure that the processing routes within the parent pipeline are connected correctly. OR not AND. All too often I see error paths not executed because the developer is expecting activity 1 AND activity 2 AND activity 3 to fail before its called. Please don’t make this same mistake. Shown below.



Finally, if you would like a better way to access the activity error details within your handler pipeline I suggest using an Azure Function. In this blog post (https://mrpaulandrew.com/2020/04/22/get-any-azure-data-factory-pipeline-activity-error-details-with-azure-functions/) I show you how to do this and return the complete activity error messages.



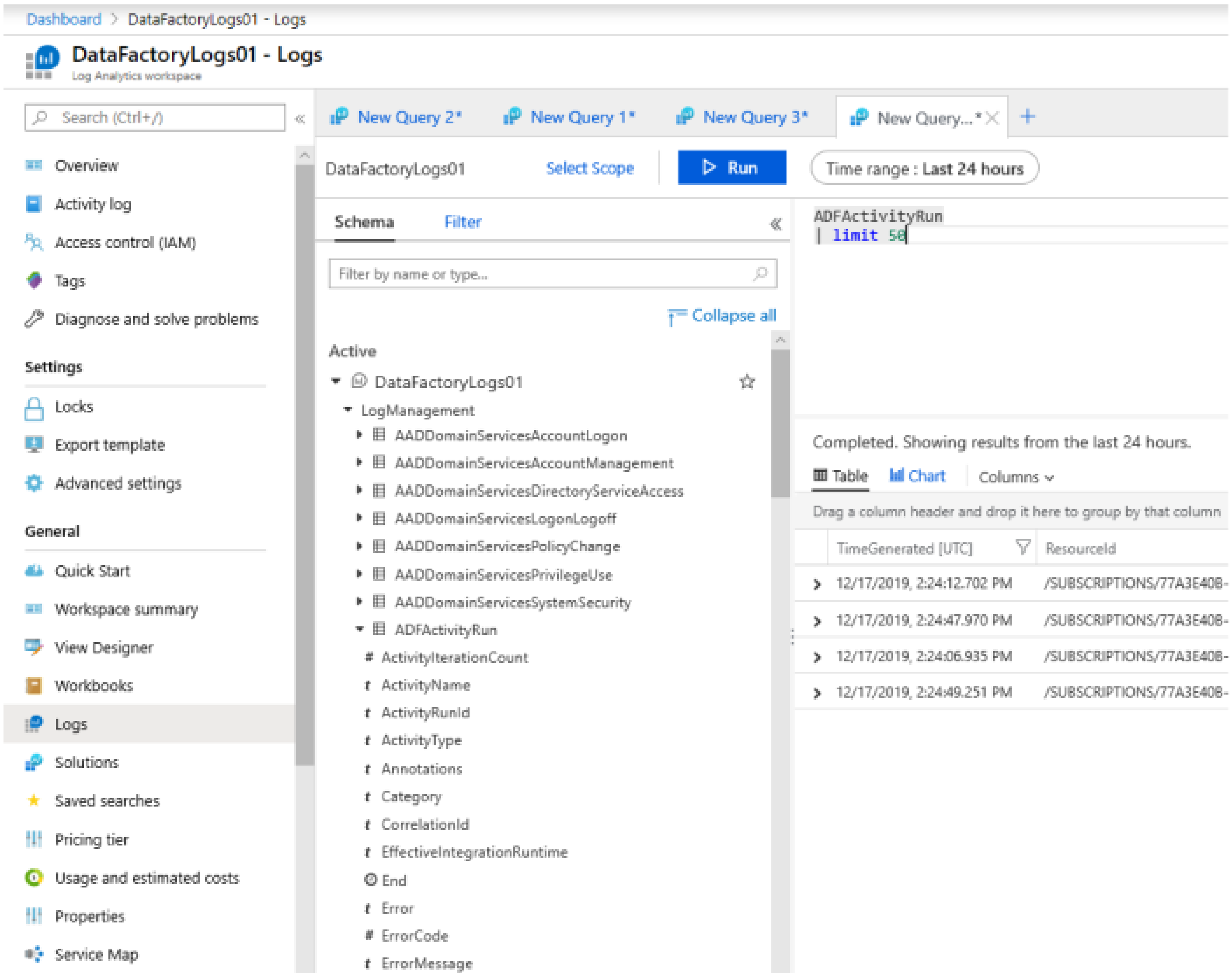
```
{
    "PipelineName": "Intention Error",
    "PipelineStatus": "Failed",
    "RunId": "0cac39e6-54b7-4f56-a4b9-96dbb80388d7",
    "ResponseCount": 3,
    "ResponseErrorCount": 2,
    "Errors": [
        {
            "ActivityName": "Call Fail Notebook",
            "ActivityType": "DatabricksNotebook",
            "ErrorCode": "3204",
            "ErrorType": "UserError",
            "ErrorMessage": "Databricks execution failed with error message: .
                                          Run page url: https://northeurope.azuredatabricks.net/?o=4757294070125047#job/18/run/1."
        },
        {
            "ActivityName": "Call Fail Procedure",
            "ActivityType": "SqlServerStoredProcedure",
            "ErrorCode": "2402",
            "ErrorType": "UserError",
            "ErrorMessage": "Execution fail against sql server. Sql error number: 245. Error Message:
                                          Conversion failed when converting the varchar value 'Force error.' to data type int."
        }
    ]
}
```

(https://mrpaulandrew.files.wordpress.com/2020/07/adf-get-activity-error-with-function.png)

Monitoring via Log Analytics

Like most Azure Resources we have the ability via the 'Diagnostic Settings' to output telemetry to Log Analytics. The out-of-box monitoring area within Data Factory is handy, but it doesn't deal with any complexity. Having the metrics going to Log Analytics as well is a must have for all good factories. The experience is far richer and allows operational dashboards to be created for any/all Data Factory's.

Screen snippet of the custom query builder shown below, click to enlarge.



```
ADFActivityRun
| limit 50
```

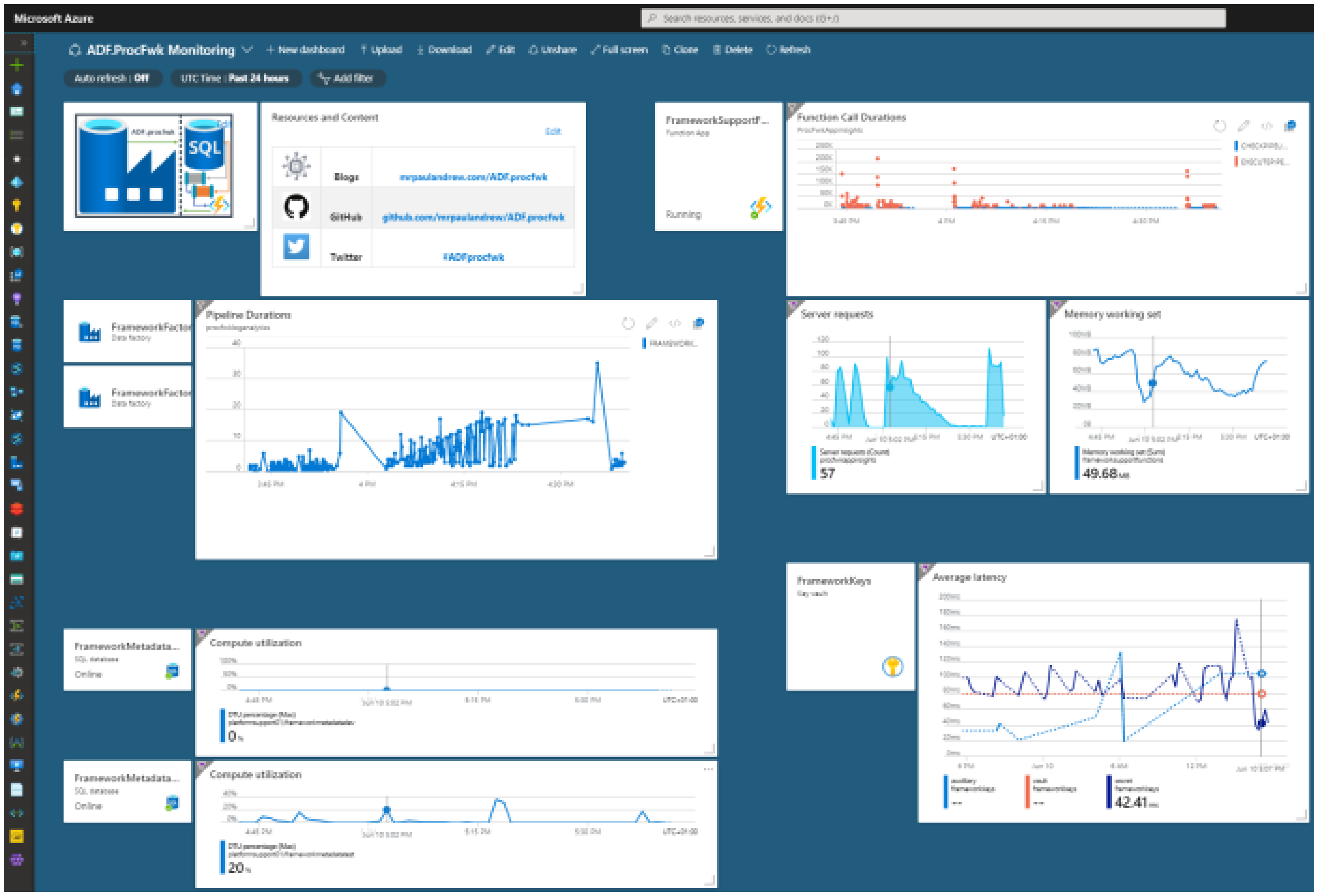
Completed. Showing results from the last 24 hours.

TimeGenerated [UTC]	ResourceId
12/17/2019, 2:24:12.702 PM	/SUBSCRIPTIONS/77A3E408-
12/17/2019, 2:24:47.970 PM	/SUBSCRIPTIONS/77A3E408-
12/17/2019, 2:24:06.935 PM	/SUBSCRIPTIONS/77A3E408-
12/17/2019, 2:24:49.251 PM	/SUBSCRIPTIONS/77A3E408-

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-output-to-log-analytics.png)

When you have multiple Data Factory's going to the same Log Analytics instance break out the Kusto queries to return useful information for all your orchestrators and pin the details to a shareable Azure Portal dashboard. For example:

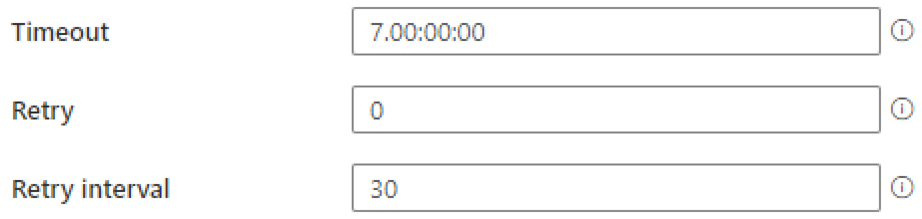
```
ADFPipelineRun
| project TimeGenerated, Start, End, ['DataFactory'] = substring(ResourceId, 121,
100), Status, PipelineName , Parameters, ["RunDuration"] = datetime_diff('Minute',
End, Start)
| where TimeGenerated > ago(1h) and Status !in ('InProgress','Queued')
```



(https://mrpaulandrew.files.wordpress.com/2020/06/portal-screen-shot-2.png)

Timeout & Retry

Almost every Activity within Data Factory has the following three settings (timeout, retry, retry interval) which fall under the **policy** section if viewing the underlying JSON:



Timeout	7.00:00:00
Retry	0
Retry interval	30

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-timeout-settings.png)

The screen shot above also shows the default values. Look carefully.

Awareness needs to be raised here that these default values cannot and should not be left in place when deploying Data Factory to production. A default timeout value of 7 days is huge and most will read this value assuming hours, not days! If a Copy activity stalls or gets stuck you'll be waiting a very long time for the pipeline failure alert to come in. Hopefully the problem and solution here is obvious. Change this at the point of deployment with different values per environment and per activity operation. Don't wait 7 days for a failure to be raised.

Credit where its due, I hadn't considered timeouts to be a problem in Data Factory until very recently when the Altius managed services team made a strong case for them to be updated across every Data Factory instance.

Cosmetic Stuff

Naming Conventions

Hopefully we all understand the purpose of good naming conventions for any resource. However, when applied to Data Factory I believe this is even more important given the expected umbrella service status ADF normally has within a wider solution. Firstly, we need to be aware of the rules enforced by Microsoft for different components, here:

https://docs.microsoft.com/en-us/azure/data-factory/naming-rules (https://docs.microsoft.com/en-us/azure/data-factory/naming-rules)

Unfortunately there are some inconsistencies to be aware of between components and what characters can/can't be used. Once considered we can label things as we see fit. Ideally without being too cryptic and while still maintaining a degree of human readability.

For example, a linked service to an Azure Functions App, we know from the icon and the linked service type what resource is being called. So we can omit that part. As a general rule I think understanding why we have something is a better approach when naming it, rather than what it is. What can be inferred with its context.

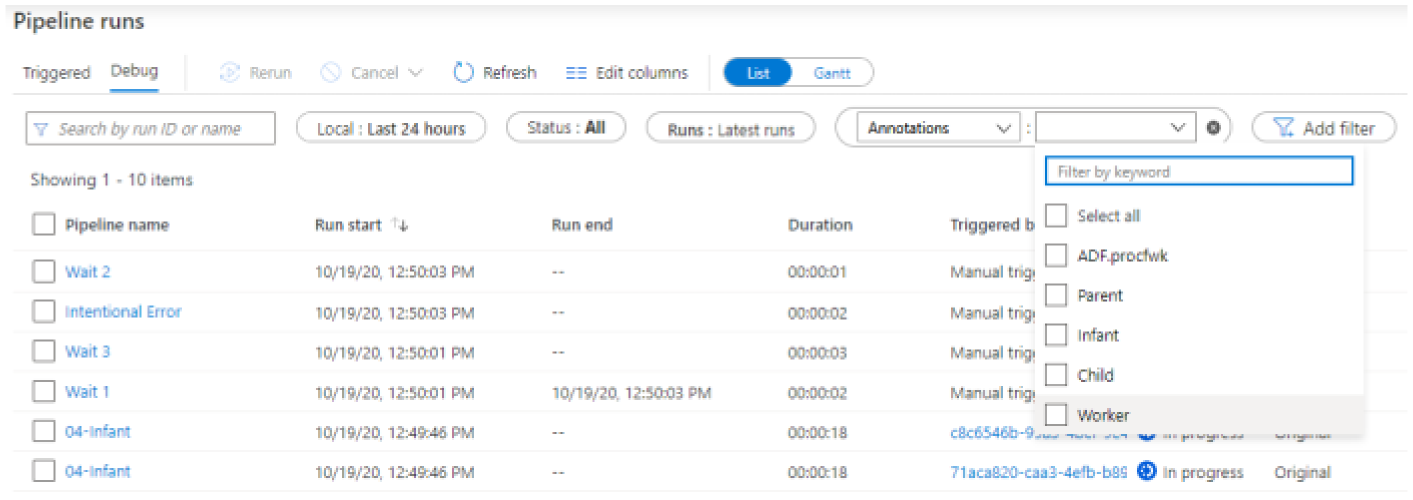
Finally, when considering components names, be mindful that when injecting expressions into things, some parts of Data Factory don't like spaces or things from names that could later break the JSON expression syntax.

Cosmetic Stuff

Annotations

All components within Data Factory now support adding annotations. This builds on the description content by adding information about 'what' your pipeline is doing as well as 'why'. If you treat annotations like tags on a YouTube video then they can be very helpful when searching for related resources, including looking around a source code repository (where ADF UI component folders aren't shown).

Another key benefit of adding annotations is that they can be used for filtering within the Data Factory monitoring screen at a pipelines level, shown below:


Pipeline runs

Triggered | Debug | Rerun | Cancel | Refresh | Edit columns | List | Gantt

Search by run ID or name | Local : Last 24 hours | Status : All | Runs : Latest runs | Annotations : | Add filter

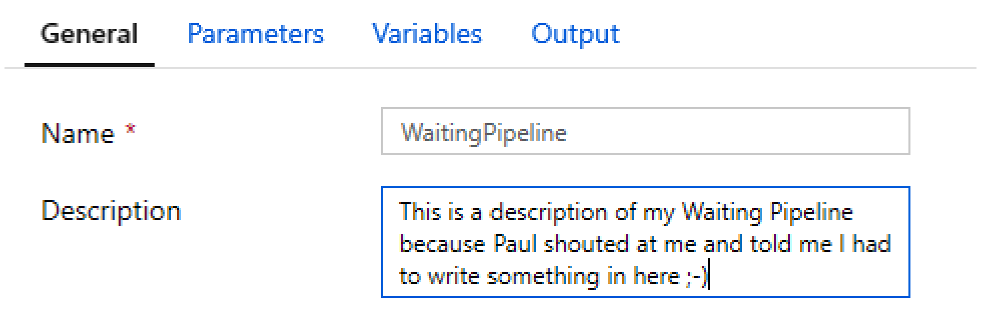
Filter by keyword
- Select all
- ADF.procfwk
- Parent
- Infant
- Child
- Worker

Showing 1 - 10 items

Pipeline name	Run start	Run end	Duration	Triggered b		
Wait 2	10/19/20, 12:50:03 PM	--	00:00:01	Manual trig		
Intentional Error	10/19/20, 12:50:03 PM	--	00:00:02	Manual trig		
Wait 3	10/19/20, 12:50:01 PM	--	00:00:03	Manual trig		
Wait 1	10/19/20, 12:50:01 PM	10/19/20, 12:50:03 PM	00:00:02	Manual trig		
04-Infant	10/19/20, 12:49:46 PM	--	00:00:18	c8c6546b-9	In progress	Original
04-Infant	10/19/20, 12:49:46 PM	--	00:00:18	71aca820-caa3-4efb-b89	In progress	Original

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-annotations.png)

Pipeline & Activity Descriptions

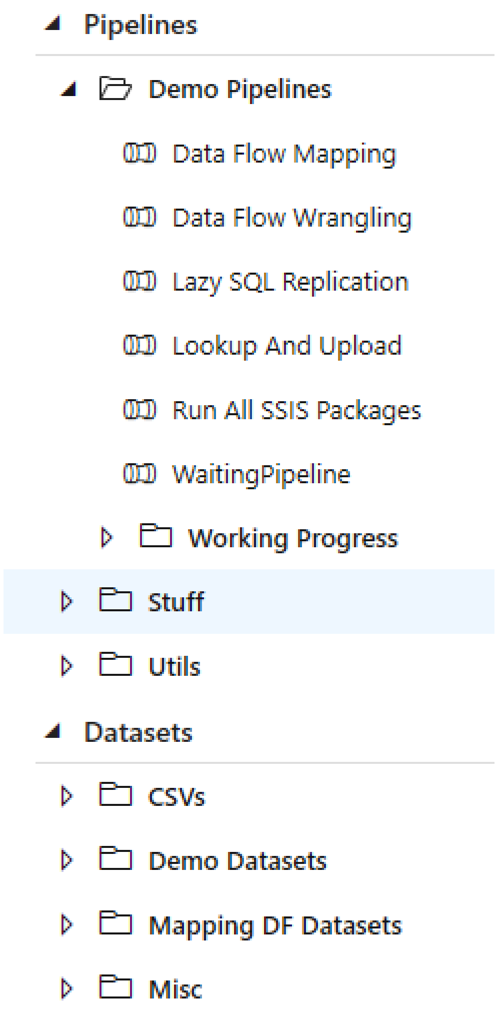

General | Parameters | Variables | Output

Name * WaitingPipeline

Description: This is a description of my Waiting Pipeline because Paul shouted at me and told me I had to write something in here ;-)

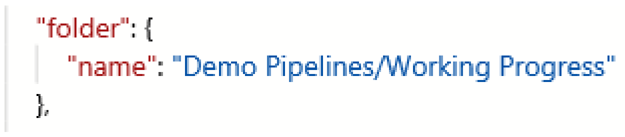
(https://mrpaulandrew.files.wordpress.com/2019/12/pipeline-description.png)Every Pipeline and Activity within Data Factory has a none mandatory description field. I want to encourage all of us to start making better use of it. When writing any other code we typically add comments to things to offer others an insight into our original thinking or the reasons behind doing something. I want to see these description fields used in ADF in the same way. A good naming convention gets us partly there with this understanding, now let's enrich our Data Factory's with descriptions too. Again, explaining why and how we did something. In a this blog post (https://mrpaulandrew.com/2019/12/19/summarise-my-azure-data-factory-arm-template-using-t-sql/) I show you how to parse the JSON from a given Data Factory ARM template, extract the description values and make the service a little more self documenting.

Factory Component Folders



(https://mrpaulandrew.files.wordpress.com/2019/12/adf-folders.png)Folders and sub-folders are such a great way to organise our Data Factory components, we should all be using them to help ease of navigation. Be warned though, these folders are only used when working within the Data Factory portal UI. They are not reflected in the structure of our source code repository or in the monitoring view.

Adding components to folders is a very simple drag and drop exercise or can be done in bulk if you want to attack the underlying JSON directly. Subfolders get applied using a forward slash, just like other file paths.



```
"folder": {
    "name": "Demo Pipelines/Working Progress"
},
```

(https://mrpaulandrew.files.wordpress.com/2019/12/adf-folders-json.png)

Also be warned, if developers working in separate code branches move things affecting or changing the same folders you'll get conflicts in the code just like other resources. Or in some cases I've seen duplicate folders created where the removal of a folder couldn't naturally happen in a pull request. That said, I recommend organising your folders early on in the setup of your Data Factory. Typically for customers I would name folders according to the business processes they relate to.

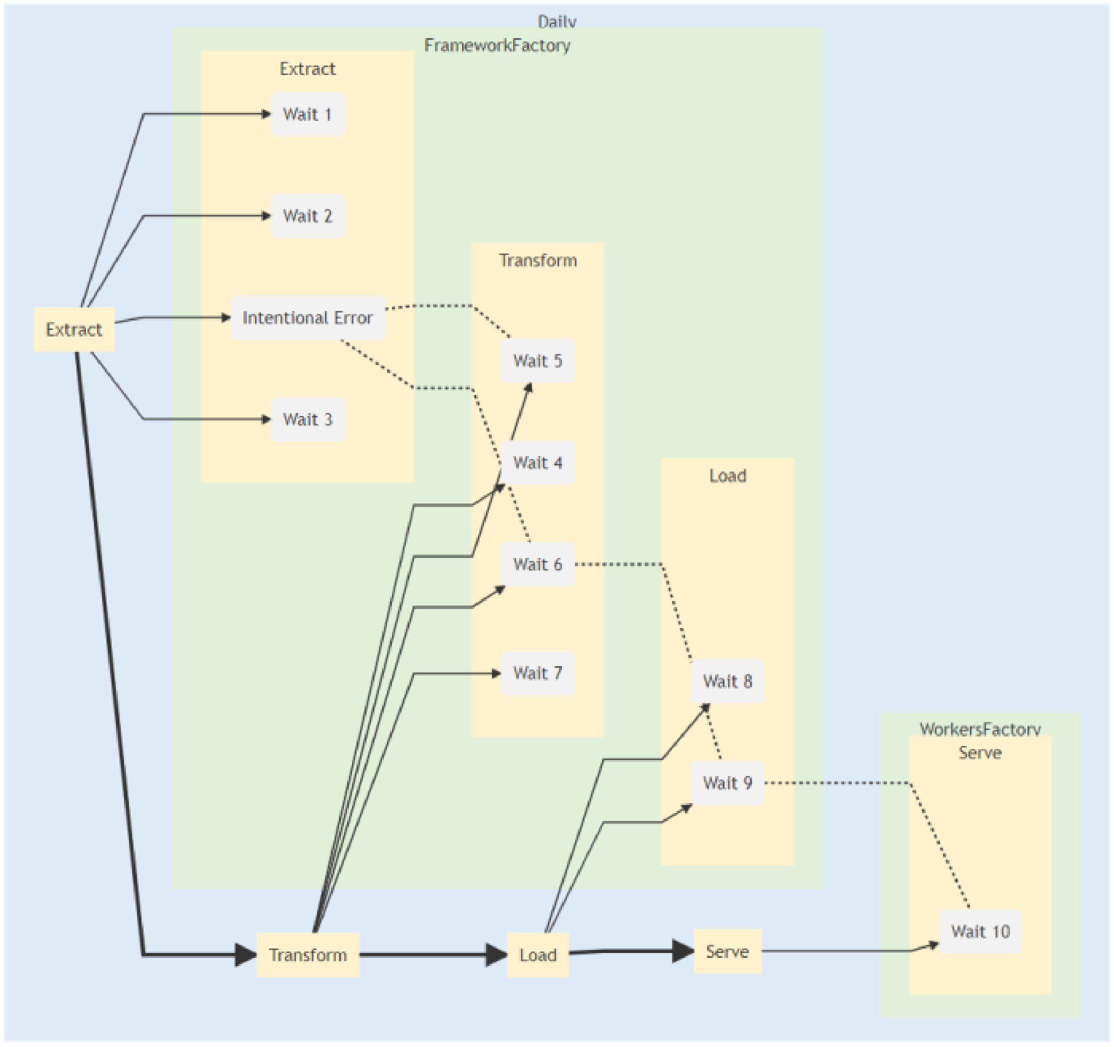
Documentation

Pipeline Lineage

Do I really need to call this out as a best practice?? Every good Data Factory should be documented. As a minimum we need somewhere to capture the business process dependencies of our Data Factory pipelines. We can't see this easily when looking at a portal of folders and triggers, trying to work out what goes first and what goes upstream of a new process. I use Visio a lot and this seems to be the perfect place to create (what I'm going to call) our Data Factory Pipeline Maps. Maps of how all our orchestration hang together. Furthermore, if we created this in Data Factory the layout of the child pipelines can't be saved, so its much easier to visualise in Visio.

If you aren't a Visio fan however and you have some metadata to support your pipeline execution chain. Try something like the below auto generated data lineage diagram, created from metadata to produce the markdown. Blogged about here:

Using Mermaid to Create a ProcFwk Pipeline Lineage Diagram (https://mrpaulandrew.com/2021/07/29/using-mermaid-to-create-a-procfwk-pipeline-lineage-diagram/)



Visio Stencils

Architecture diagrams are one thing, but what about diagrams or mock ups of out pipelines.

Instead of building a set of pipelines activities or the internals of a Data Flow directly in Data Factory using Visio can be a handy offline development experience.