WELCOME TO THE TECHNICAL COMMUNITY BLOG OF PAUL ANDREW

Data Platform Principal Consultant & Solution Architect

Best Practices for Implementing Azure Data Factory

Posted on December 18, 2019July 15, 2020
Blog post updated 15th July 2020

(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 5 years of working with ADF I think its time to start suggesting what I'd expect to see in any good Data Factory, 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. As an overview, I'm going to cover the following points:

- Environment Setup & Developer Debugging *(updated)*
- Deployments *(new)*
- Automated Testing *(new)*
- Naming Conventions
- Pipeline Hierarchies
- Pipeline & Activity Descriptions *(updated)*
- Factory Component Folders
- Linked Service Security via Azure Key Vault
- Dynamic Linked Services *(new)*
- Generic Datasets *(updated)*
- Metadata Driven Processing *(updated)*
- Parallel Execution
- Hosted Integration Runtimes
- Azure Integration Runtimes
- Wider Platform Orchestration
- Custom Error Handler Paths *(updated)*
- Monitoring via Log Analytics *(updated)*

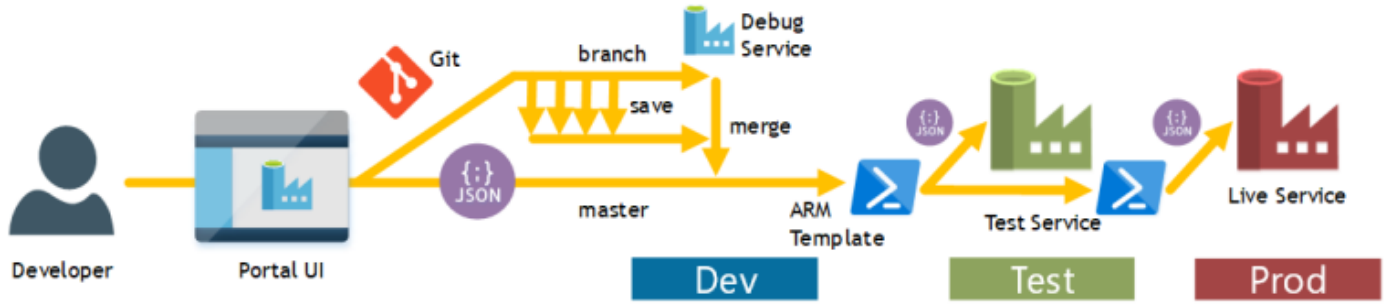
- Service Limitations *(new)*
- Using Templates
- Documentation

Let's start, my ADF best practices:

Environment Setup & Developer Debugging

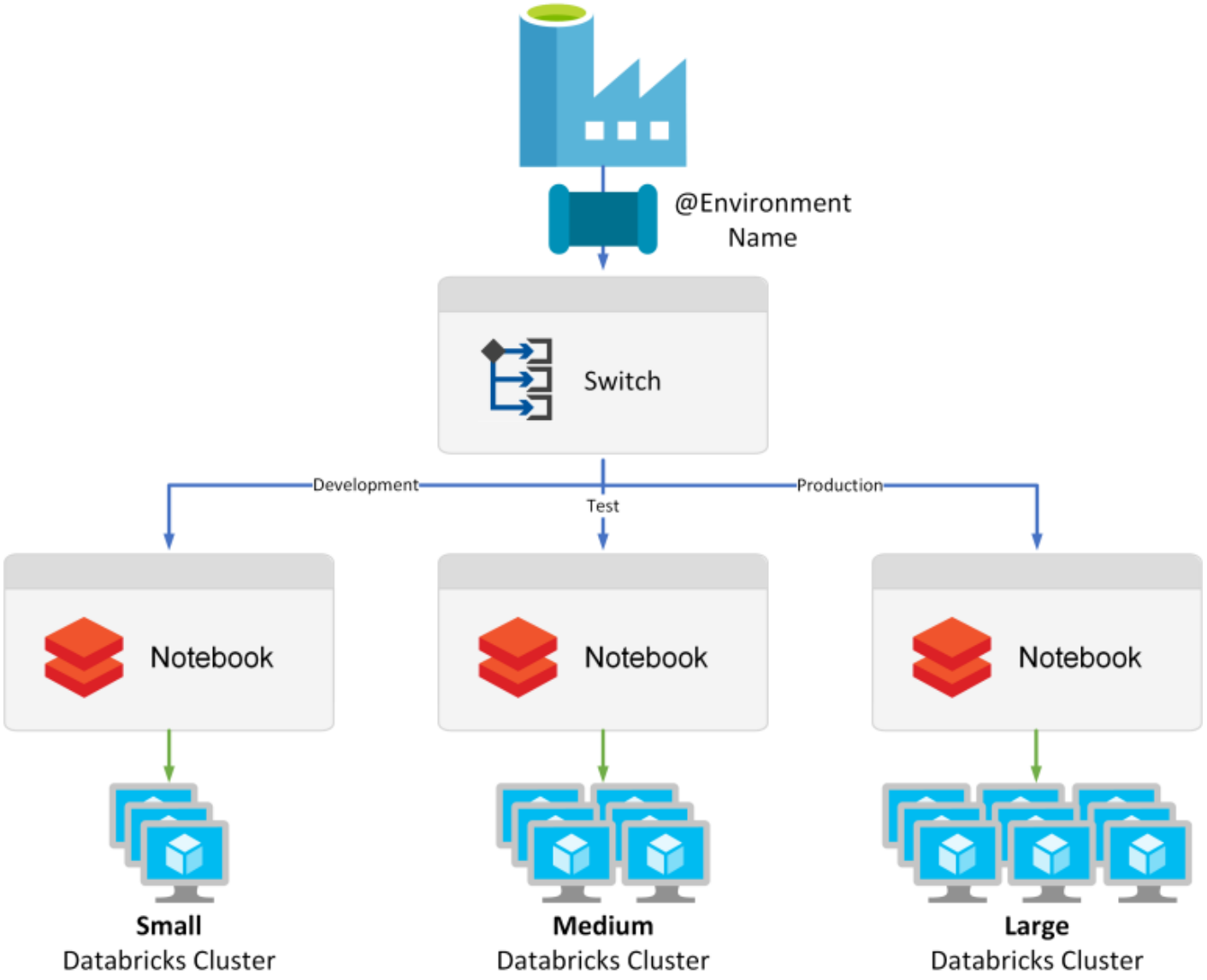
Having a clean separation of resources for development, testing and production. Obvious for any solution, but when applying this to ADF, I'd expect to see the development service connected to source control as a minimum. Using Azure DevOps or GitHub doesn't matter, although authentication against Azure 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-releases.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:



(https://mrpaulandrew.files.wordpress.com/2020/01/adf-switch-vs-db-notebook.png)

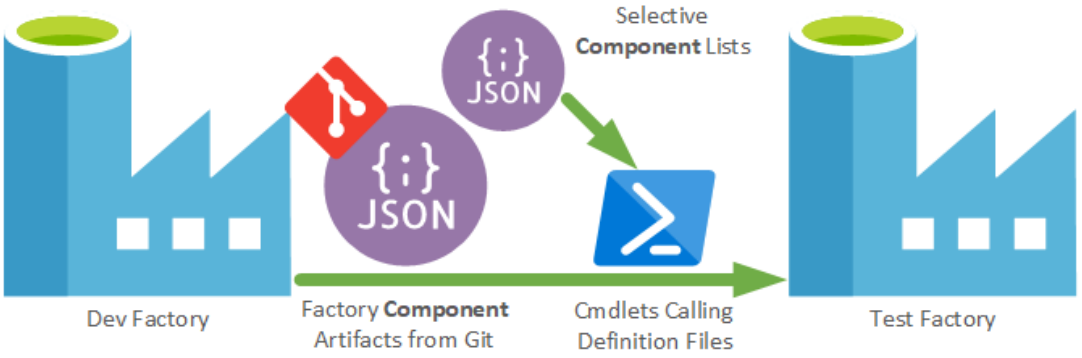
This is probably a special case and nesting activities via a 'Switch' does come with some drawbacks. This is not a best practice, but an alternative approach you might want to consider. I blogged about this in more detail here (https://mrpaulandrew.com/2020/01/22/using-the-azure-data-factory-switch-activity/).

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

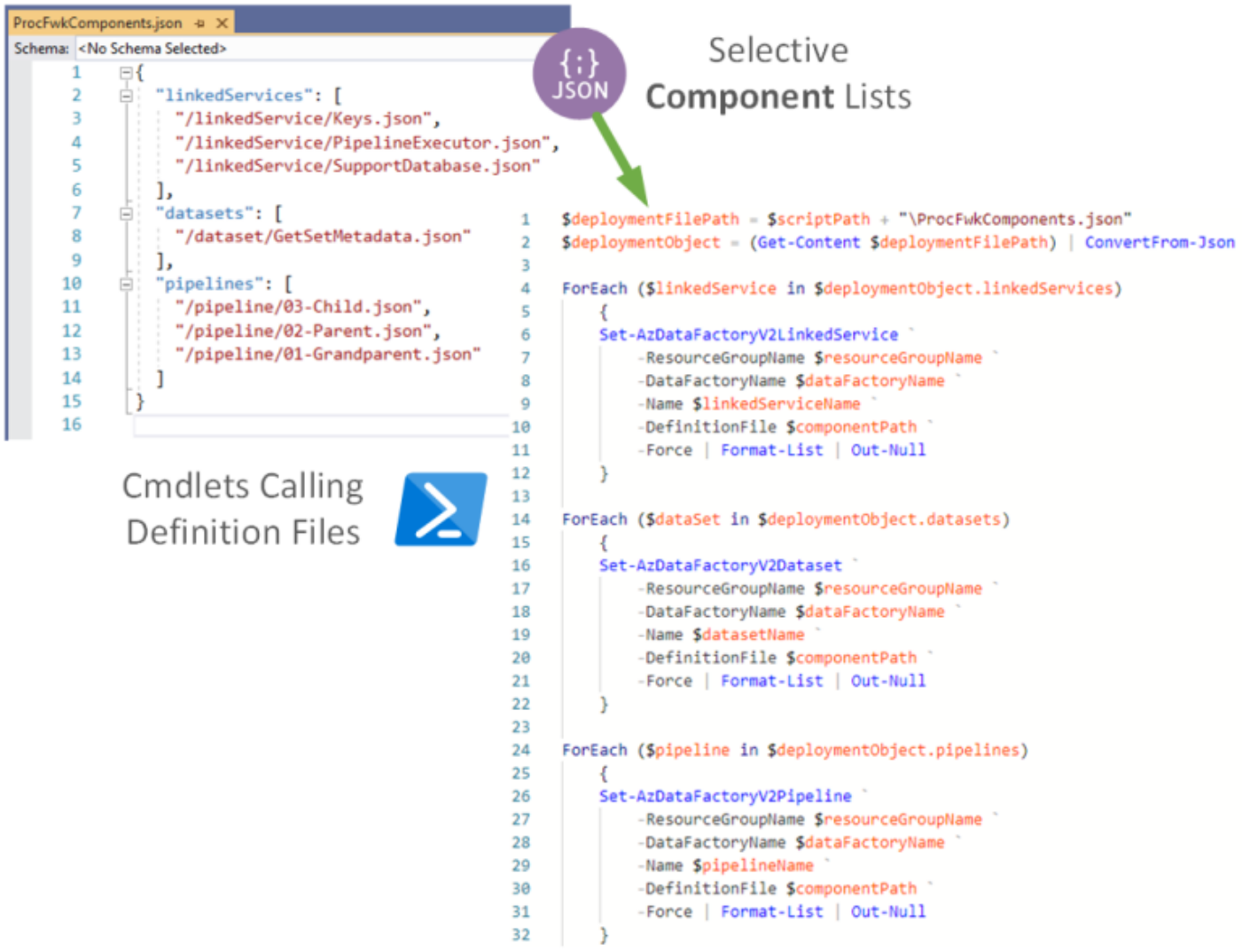
dynamically changing any parts of the JSON at deployment time. But, is does mean you have to manually handle component dependencies and removals, if you have any. A quick visual of the approach:



(https://mrpaulandrew.files.wordpress.com/2020/04/adf-deployment-with-powershell.png)

To elaborate, the PowerShell uses the artifacts created by Data Factory in the 'normal' repository code branches (not the **adf_publish** branch). 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.



(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.

Automated Testing

(https://mrpaulandrew.files.wordpress.com/2020/07/adf-pipeline-pass-fail.png)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 using an NUnit project in Visual Studio. Check it out here (https://richardswinbank.net/tag/adftesting?do=showtag&tag=adftesting).

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.

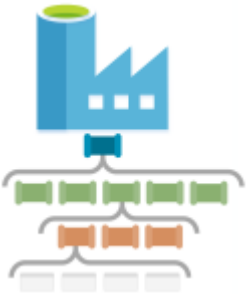
For example, a linked service to an Azure Functions App, we know from the icon and the linked service type what resource is being called. So we can omit that part. As a general rule I think understanding why we have something is a better approach when naming it, rather than what it is. What can be inferred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

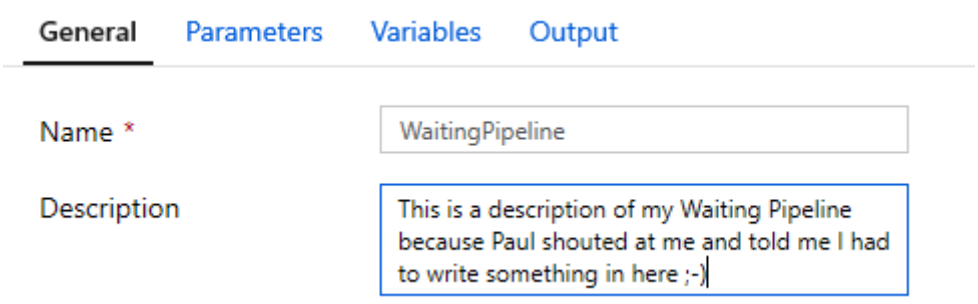
Pipeline Hierarchies

(https://mrpaulandrew.files.wordpress.com/2019/09/adf-gpc-pipelines.png)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…

- Grandparent
- Parent
- Child
- Infant



Pipeline & Activity Descriptions



(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

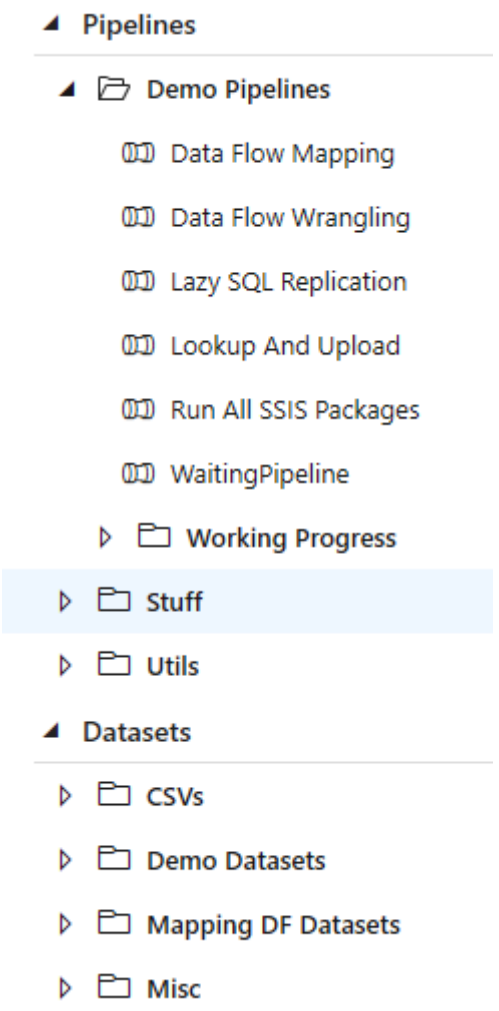
(https://mrpaulandrew.files.wordpress.com/2019/12/adf-folders.png)Folders and sub-folders are such a great way to organise our Data Factory components, we should all be using them to help ease of navigation. Be warned though, these folders are only used when working within the Data Factory portal UI.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

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.



Linked Service Security via Azure Key Vault

(https://mrpaulandrew.files.wordpress.com/2018/11/key-vault-icon.png)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 obtaining 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 Identity.

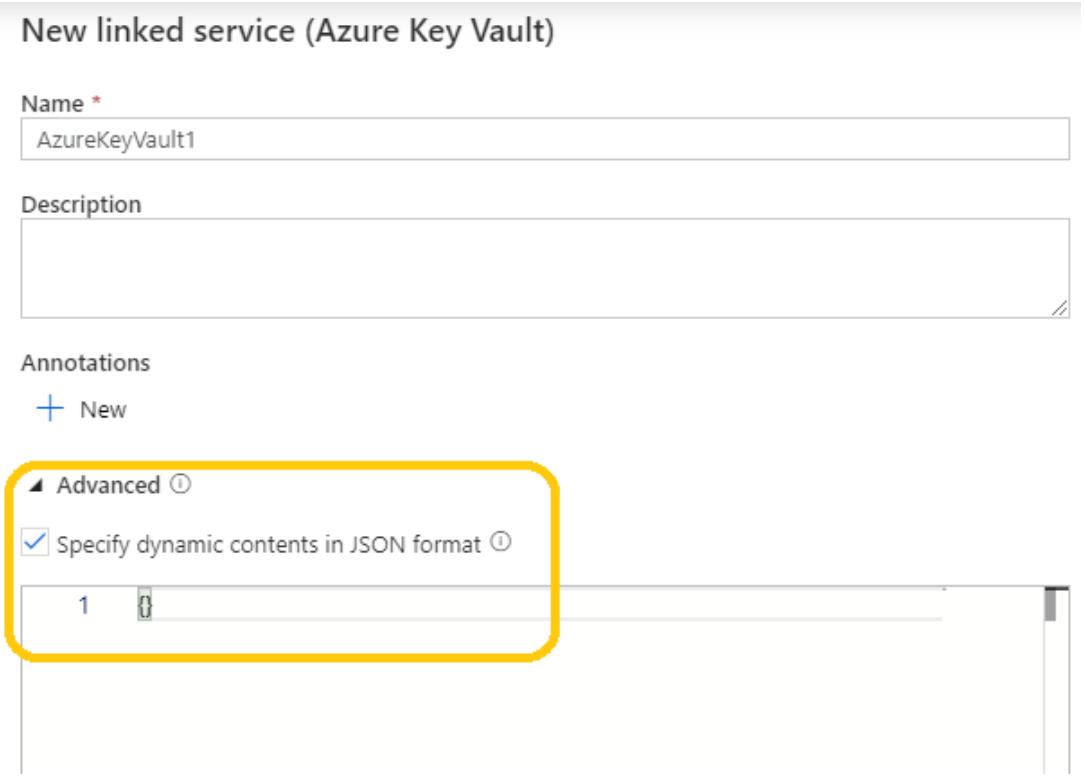
If you aren't familiar with this approach check out this Microsoft Doc pages:

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.

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/).

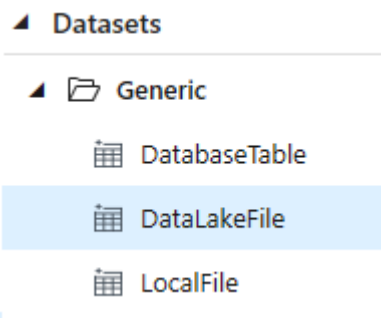


(https://mrpaulandrew.files.wordpress.com/2020/07/adf-ls-dynamic-content-kv-tick-1.png)

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.

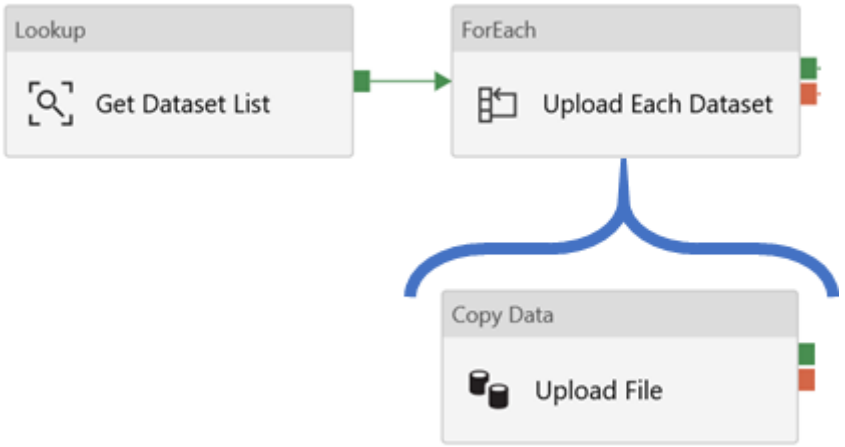
- **Location** – the file path, table location or storage container.
- **Name** – the file or table name.
- **Structure** – the attributes available provided as an array at runtime.

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.

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.

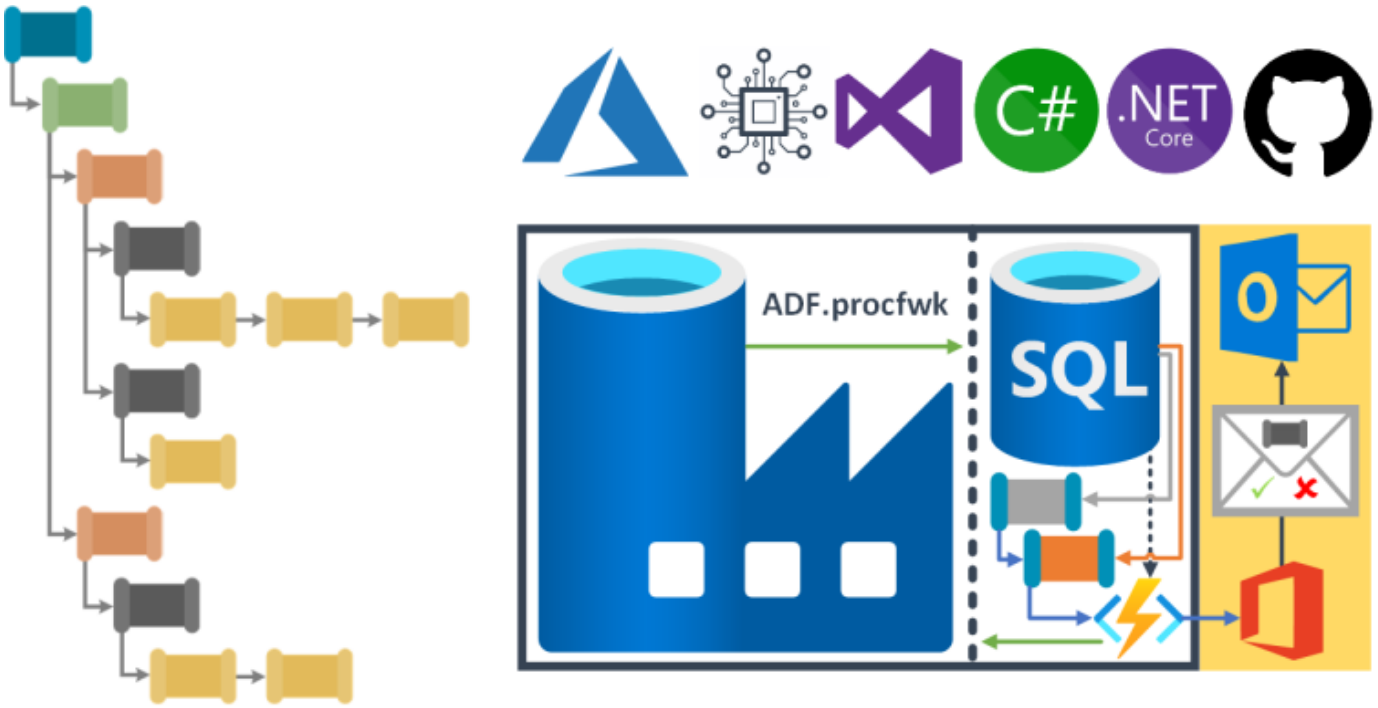
This metadata driven approach means deployments to Data Factory for new data sources are greatly reduced and only adding new values to a database table is required. The pipeline itself doesn't need to be complicated. Copying CSV files from a local file server to Data Lake Storage could be done with just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



(https://mrpaulandrew.files.wordpress.com/2019/12/metadata-driven-ingestion.png)

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

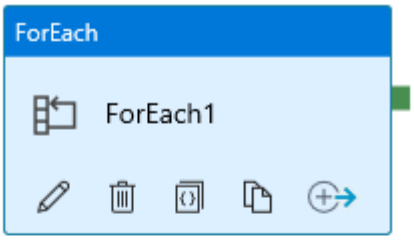
- https://mrpaulandrew.com/category/azure/data-factory/adf-procfwk/ (https://mrpaulandrew.com/category/azure/data-factory/adf-procfwk/)
- https://github.com/mrpaulandrew/ADF.procfwk (https://github.com/mrpaulandrew/ADF.procfwk)

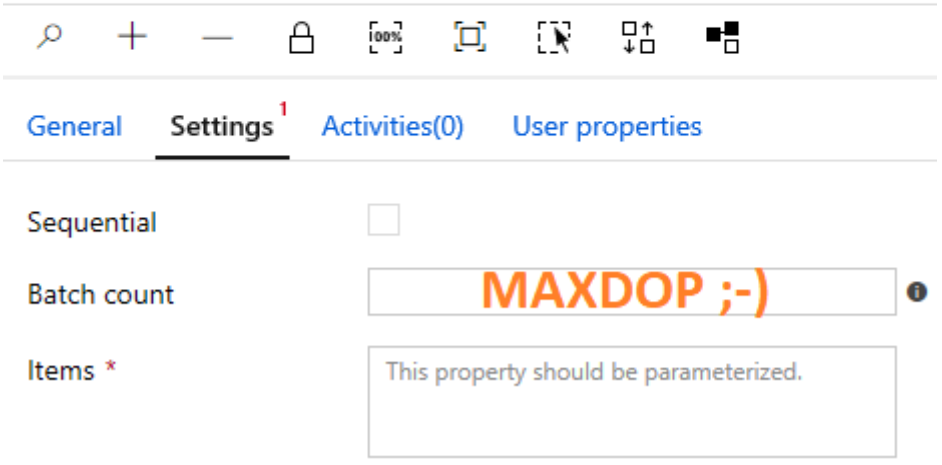


(https://mrpaulandrew.files.wordpress.com/2020/06/repo-image.png)

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.



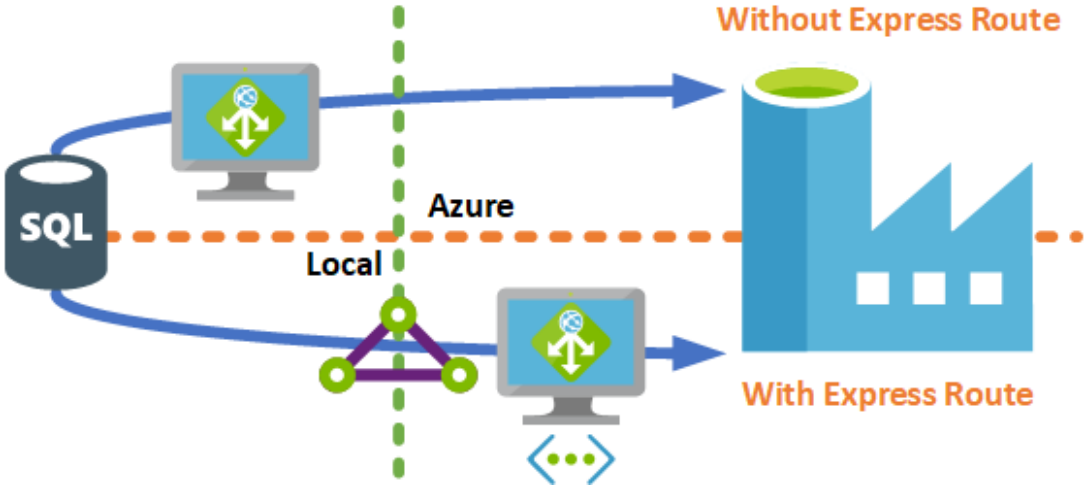


(https://mrpaulandrew.files.wordpress.com/2019/12/foreach-settings.png)
In case you weren't aware within the ForEach activity you need to use the syntax **@{item().SomeArrayValue}** to access the iteration values from the array passed as the ForEach input.

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). 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.



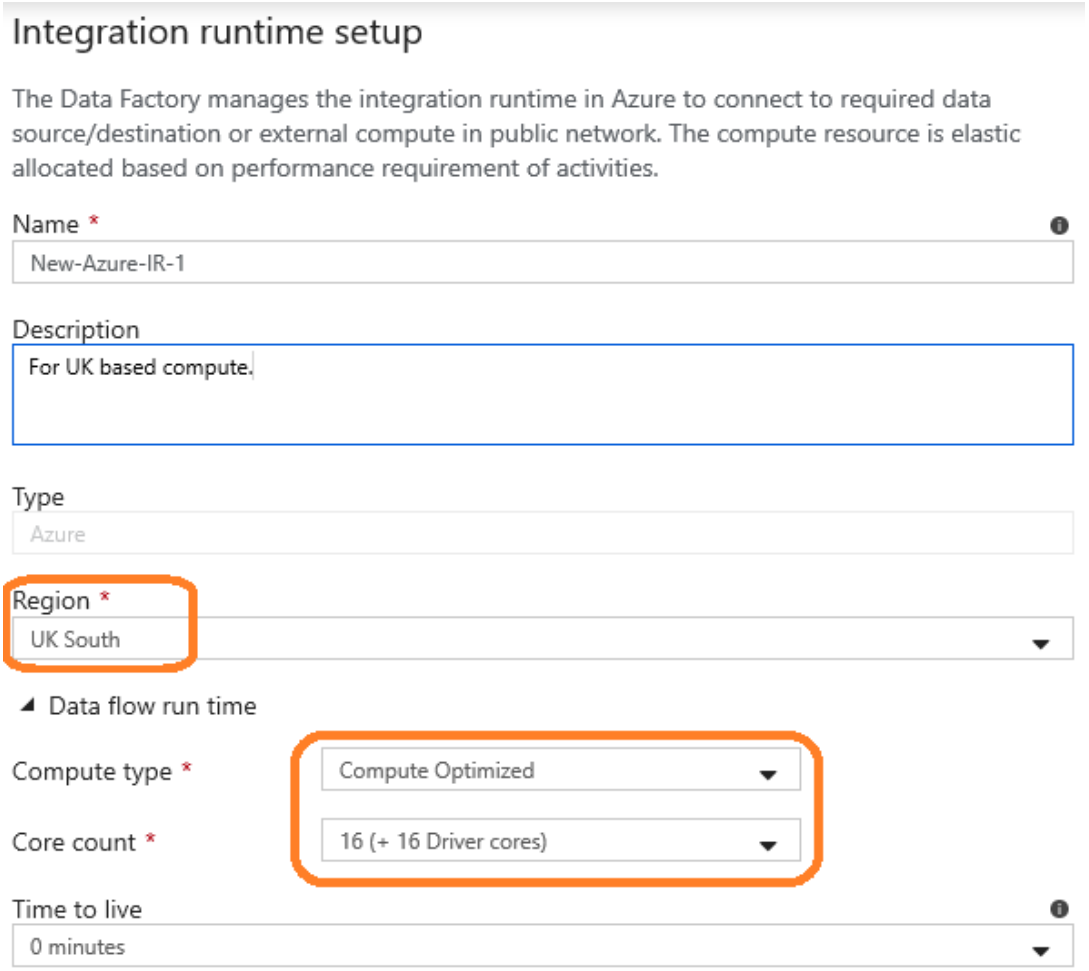
(https://mrpaulandrew.files.wordpress.com/2019/12/adf-hosted-ir-express-route-vs-not.png)

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.



(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.

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)

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.

(https://mrpaulandrew.files.wordpress.com/2019/12/error-handling-paths-in-adf.png)

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.

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

Monitoring via Log Analytics

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

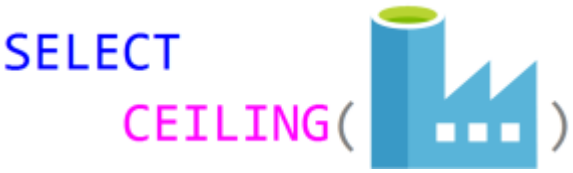
(https://mrpaulandrew.files.wordpress.com/2019/12/adf-output-to-log-analytics.png)

When you 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)

Service Limitations



(https://mrpaulandrew.files.wordpress.com/2020/01/adf-ceiling.png)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.

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.

Documentation

Lastly, for this blog post! 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. I recommend we all start creating something similar to the example below.

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

That's all folks. If you have done all of the above when implementing Azure Data Factory then I salute you 🙂

Many thanks for reading. Comments and thoughts very welcome. Defining best practices is always hard and I'm more than happy to go first and get them wrong a few times while we figure it out together 🙂

Tagged Azure Data Factory, Best Practice

Published by mrpaulandrew

Principal consultant and architect specialising in big data solutions on the Microsoft Azure cloud platform. Data engineering competencies include Azure Data Factory, Data Lake, Databricks, Stream Analytics, Event Hub, IoT Hub, Functions, Automation, Logic Apps and of course the complete SQL Server business intelligence stack. Many years' experience working within healthcare, retail and gaming verticals delivering analytics using industry leading methods and technical design patterns. STEM ambassador and very active member of the data platform community delivering training and technical sessions at conferences both nationally and internationally. Father, husband, swimmer, cyclist, runner, blood donor, geek, Lego and Star Wars fan!
View all posts by mrpaulandrew

17 thoughts on "Best Practices for Implementing Azure Data Factory"

1. **Julio** says:
   December 18, 2019 at 11:05 am
   thank you so much!!!! this post is great!!!!!!!!!

2. Reply
   Remco says:
   December 18, 2019 at 11:42 am
   Big Thanks Paul! Really useful tips!

   Reply
3. Pingback: Recommendations for Implementing Azure Data Factory – Curated SQL
4. **Vidar** says:
   December 20, 2019 at 8:38 am
   Do you have any thoughts on how to best test ADF pipelines?

   Reply
   1. **mrpaulandrew** says:
      December 20, 2019 at 10:34 am
      That's an interesting one… are you testing the ADF pipeline? Or are you actually testing whatever service the ADF pipeline has invoked? If the former and you don't care about the service called what are you even testing for?

Reply

vslatten says:
December 20, 2019 at 11:21 am
We are currently not doing any pipeline tests, just looking at how we may do it. We would mainly be interested in integration tests with the proper underlying services being called, but I guess we could also parameterize the pipelines sufficiently that we could use mock services and only test the pipeline logic, as a sort of unit test. The thinking so far is to have a separate folder in ADF for "test pipelines" that invoke other pipelines and check their output, then script the execution of the test pipelines in a CI build.

BTW., Thanks for the very nice writeup in the original article!

5. **Nwyra** says:
December 20, 2019 at 1:50 pm
Excellent post and great timing as I'm currently getting stuck into ADF. We are following most of the above, but will definitely be changing a few bits after reading you're post. One point we are unsure of is if we should be setting up a Data Factory per business process or one mega Factory and use the folders to separate the objects.

Current thoughts are to create a factory for each distinct area (so one for Data Warehouse, one for External File delivery etc.) then one extra factory just containing the integration runtimes to our on-prem data that are shared to each factory when needed. From a CI pipeline point of view then, with separate factories one release doesn't hold up the other. We don't see us having more than about 10 distinct areas overall.

Be great to hear your thoughts on best practice for this point. Thanks again for the great post.

6. Reply
Marc Jellinek says:
December 30, 2019 at 3:27 am
I'm working on what I hope will be a best-practices reference implementation of Data Factory pipelines. Please check out https://github.com/marc-jellinek/AzureDataFactoryDemo_GenericSqlSink if you have a minute. I'm calling the project Throwing Mud on the Wall.

Reply

7. Pingback: My Script for Peer Reviewing Code – Welcome to the Technical Community Blog of
8. Paul Andrew
Robert Koechig says:
March 9, 2020 at 1:06 am
Great article – love the scale up / down steps – great tip!

9. Reply
Matthew Darwin says:
March 13, 2020 at 10:20 am
Hi Paul, great article! As someone relatively new to Data Factory, but familiar with other components within Azure and previous lengthy experience with SSIS, I wanted to as a couple of questions:-

1. Would (and if so when would) you ever recommend splitting into multiple Data Factories as opposed to having multiple pipelines within the same Data Factory? In my head I'm currently seeing a Data Factory as analagous to a project within SSIS. From a deployment point of view, in SSIS having a "too big" project started making deployments and testing a little unwieldy with the volume of things being deployed; such as having to ensure that certain jobs were not running

during the deployment and so on. Would this be the case with deploying the ARM template for DF? I'm also thinking of the security aspects, as I'm assuming RBAC is granted at the factory level?

2. Does DF suffer from the same sort of meta data issues that SSIS did? E.G. when an underlying table had a column that was not used in a data flow changed, you still needed to refresh the metadata within SSIS even though effectively no changes were being made.

3. From a code review/pull request perspective, how easy is it to look at changes within the ARM template, or are they sometimes numerous and unintelligible as with SSIS and require you to look at it in the UI? For SSIS collaborative working has always been a bit of a pain due to the shared xml structure.

4. I was mentioned above around testing, what frameworks (if any) are you currently using?

Thanks in advance if you get time to answer any of that, turned into more text than I was anticipating!

Reply

mrpaulandrew says:
March 30, 2020 at 10:01 am
Hi Matthew, thanks for the comments, maybe let's have a chat about your points rather than me replying here. Drop me an email and we can arrange something. Cheers Paul

Reply

Omar Khan says:
May 10, 2020 at 11:26 pm
Great Article Paul, I have same questions as Matthew Darwin, was wondering if you have replied to them.

2. **Adrian DeFazio** says:
May 21, 2020 at 4:39 pm
It would definitely be good to hear an opinion on question number 1. I believe a lot of developers of ADF will struggle with this best practice. I can see a Data Factory becoming hard to maintain if you have multiple pipelines for different processes in the same Data Factory. It sounds like you can organize by using folders, but for maintainability it could get difficult pretty quickly.

3. **mrpaulandrew** says:
May 22, 2020 at 9:12 am
Hi All

Sorry for the delayed reply.

To point 1 in Matthew's comment above.

There are a few things to think about here:

Firstly, I would consider using multiple Data Factory's if we wanted to separate business processes. E.g. Finance, Sales, HR. Another situation might be for operations and having resources in multiple Azure subscriptions for the purpose of easier inter-departmental charging on Azure consumption costs. Finally, data regulations could be a factor. For example, resources are restricted to a particular Azure region. You can overcome this using

region-specific Azure Integration Runtimes with ADF, but that adds a management overhead to deployments of IR's and aligning specific (copy) Activities to those dedicated IR's. Therefore, separate factories might make more sense.

So, in summary, 3 reasons to do it… Business processes. Azure charging. Regulatory data restrictions.

What I would not do is separate Data Factory's for the deployment reasons (like big SSIS projects). That said, it is also worth pointing out that I rarely use the ADF ARM templates to deployment Data Factory. Typically, we use the PowerShell cmdlets and use the JSON files (from your default code branch, not 'adf_publish') as definitions to feed the PowerShell cmdlets at an ADF component level. PowerShell as granular cmdlets for each ADF component (Datasets, Linked Services, Pipelines etc). This gives much more control and means releases can be much smaller.

Final thoughts, around security and reusing Linked Services. ADF does not currently offer any lower level granular security roles beyond the existing Azure management plane. Therefore, once you have access to ADF, you have access to all its Linked Service connections. Maybe this is a reason to separate resources and in-line with my first point about business processes.

Hope this helps inform your decisions.

Cheers
Paul

10. **enguerran** says:
April 6, 2020 at 11:24 am
Hi Paul, Great article, i was wondering. I'm convinced by the github integration only on the dev env. (and by many other good practices you describe. I was wondering though : that means the triggers have to be scheduled the same way on all the environments ?

Reply

mrpaulandrew says:
April 7, 2020 at 8:59 am
Hi, yes great point, in this situation I would do Data Factory deployments using PowerShell which gives you much more control over things like triggers and pipeline parameters that aren't exposed in the ARM template. Rather than using a complete ARM template, use each JSON file added to the repo 'master' branch as the definition file for the respective PowerShell cmdlet.

For example:
…\Git\pipeline\Pipeline1.json. Then in PowerShell use Set-AzDataFactoryV2Pipeline -DefinitionFile $componentPath.

For a trigger, you will also need to Stop it before doing the deployment. So, something like…

```
$currentTrigger = Get-AzDataFactoryV2Trigger
if($currentTrigger -ne $null)
Stop-AzDataFactoryV2Trigger
Set-AzDataFactoryV2Trigger
Start-AzDataFactoryV2Trigger
```

Hope this helps.

Reply

This site uses Akismet to reduce spam. Learn how your comment data is processed.

POWERED BY WORDPRESS.COM.