Azure Data Factory Parameterized Linked Services

keepsecure.ca/blog/azure-data-factory-parameterized-linked-services

By Shamir Charania on July 18, 2020

In Azure Data Factory (ADF), managing linked services can be a bit tricky. It is often quite the balancing act to figure out who should manage them, how they should be managed across environments, and how to audit/monitor them from a security perspective. It can get especially interesting to manage data factories that contain multiple projects or use cases. The question becomes where is it best to manage and audit the linked services configuration.

If you are using Azure resource manager (ARM) templates to manage your ADF, you might initially think that building individual linked services for all the uses cases and then making use of template parameters is the right way to go. This could result in a potentially large amount of linked services to manage and enforce configuration on.

An alternate way to do things is to look at parameterized linked services. In this setup, datasets (when configured) are prompted for additional information relating to the target linked service. This allows you to reduce the number of linked services, pushing up configuration to the dataset layer. In your environment, this may or may not be easier to audit/maintain/secure then a myriad of linked service connections individually connected.

The goal of this post is to walk through how to set up parameterized linked services. You can read the official documentation here. In this example, we walk through how to do this for an Azure storage account.

When you create a linked service for an Azure storage account, the following code gets automatically created.

```
{
   "name": "AzureBlobStorage1",
   "properties": {
      "annotations": [],
      "type": "AzureBlobStorage",
      "typeProperties": {
         "connectionString": {
            "type": "AzureKeyVaultSecret",
            "store": {
               "referenceName": "AzureKeyVault1",
               "type": "LinkedServiceReference"
            },
            "secretName": "storageconnstring"
         }
      }
   }
}
```

As you can see above, I've opted to store my connection string in key vault. When you use the standard configuration above, you get the following view while creating a dataset.

Let's now add a parameter so that the key name used to store the connection string can be inserted from the dataset, rather than hardcoded in the linked service.

```
{
   "name": "AzureBlobStorage2",
   "properties": {
      "annotations": [],
      "type": "AzureBlobStorage",
      "typeProperties": {
         "connectionString": {
            "type": "AzureKeyVaultSecret",
            "store": {
               "referenceName": "AzureKeyVault1",
               "type": "LinkedServiceReference"
            },
            "secretName": "@{linkedService().keyVaultSecretName}"
         }
      },
      "parameters": {
         "keyVaultSecretName": {
            "type": "String"
         }
      }
   }
}
```

Here is what the dataset creation looks like now:

Conclusion

While this approach might not make a ton of sense for changing key names dynamically on a storage account, it might come in handy for other linked services, such as Azure Databricks, so that different datasets and pipelines can specify different cluster Ids. This is of course dependant on how you configure your overall ETL process, as now that the linked service is parameterized, it does not show up in the ARM template parameters.

Share This Article

Comments