# APS DR framework- Implementation using ADF pipelines

# OBJECTIVE

The framework presents the Disaster Recovery (DR) solution architecture for Microsoft’s Analytics Platform System (APS). While there are several [DR Architectures available for APS](https://github.com/aisley/Hybrid_DR_APS_SQLDW); this IP focuses on Dual Load approach implementation.

The Dual Load solution performs step-by-step tracking of each source data file and ensures all files are loaded, without compromising on data integrity, into both the Primary & Secondary APS systems with an ETL workflow automation done using Azure Data Factory (ADF).

ADF, being managed cloud service, provides in-built capabilities for easy monitoring and logging with not only robust access to the details of every process or workflow step in your Data Factory but out of the box alerting in the event that something does go awry. ADF additionally provides convenient diagram views within Azure Portal to analyze workflows. ADF is also pretty flexible in setting up custom workflow schedules.

DUAL LOAD DR FRAMEWORK

Dual Load framework is accomplished by following processes:

1. A process that pulls all the source system data to be processed in a central location. When dual loading two systems, the dual ETL process will not pull the data at the same rate or at exactly the same time. Therefore, having a central process that pulls the data and generates a flat file that both systems use, the potential of one system being different than the other will be lessened. Using flat files allows easy method to ensure both the Primary and Secondary PDW systems stay 100% in sync.
2. A central Monitoring or control DB process is used to keep track of when each flat file gets loaded on each of the APS systems.
3. The dual load process implemented with 3 separate workflows or base pipelines:
   1. **Init Workflow Pipeline** begins the dual load process by retrieving a list of flat files stored in a directory and recording the location and filename in the control db.
   2. **Load Process Workflow Pipeline** reads the central control DB and determines the flat files that need to be loaded into the PDW. Should one of the PDW go offline, no steps will need to be taken. When the PDW comes back online, the workflow will process the files that have not been processed for the given PDW.
   3. **Archive Workflow Pipeline** reviews the control tables to determine what flat files have been processed by both systems. If the file has been processed by both systems, the flat file will be moved to an archive location. If the file has only been processed by one system, the files will remain on the disk until the other system can process the file.

DESIGN CHALLENGES using ADF

DR framework requires dual load workflows not just limited to pre-defined source data but be flexible enough to process/load any new flat files from source system with every extract. Also in addition to data flow, framework workflows required to pass the variables to subsequent steps.

For this to be accomplished using ADF technology- ADF components including pipelines, activities and datasets need to be created specific to each input data files.

But, ADF in current form lacks out of the box support for: (1) dynamic datasets generated based on input data, (2) looping constructs for creating N pipelines needed for processing N data files, or (3) passing state or variables across activities. Per ADF product team- "These features (looping, state passing, etc) are in design phases as part of us expanding the ADF app model. Too early to say yet though re: release dates, etc. "

**Solution:** To enable ADF ingesting new data files from source with every extract; N processing Pipelines needed to be created dynamically (N being dynamic number based on # of new files at source), and then being able to intervene in the flow for passing the state to subsequent activities within a pipeline

Various ADF creation methods were evaluated. Creating ADF thru Azure Portal, or with Azure PowerShell, or using Visual Studio ADF plugins all had limitations as they work primarily with pre-defined input datasets.

It needed a custom approach to meet these special requirements. The dynamicity of the solution was achieved by creating Data Factory, and all its components programmatically using Data Factory and Azure APIs. The program would initially query ETL control configurations, and source data files with every extract, and based on that generate processing pipelines via ADF APIs.

SETUP

* You will need an Azure account, access to the new Azure Portal and Visual Studio installed and configured on your machine. The framework uses an Azure SQL database, Azure Blob storage and of course Azure Data Factory.
* Azure SQL database for Control DB can be created directly via Azure Portal.
* All DDLs & stored procedures (available in VS project) can be loaded on Control DB by connecting thru SQL Server management Studio. Sample DMLs are also provided for sample data loading.
* Azure Storage account with folders for storing Source, ToBeProcessed & Archived data files can be created directly from Azure Portal
* Creating Data Factory:

Prerequisites

* Visual Studio 2015
* install Azure .NET SDK (from [MS download site](https://azure.microsoft.com/en-us/downloads/))
* install NuGet packages for Azure Data Factory
  + Click Tools, point to NuGet Package Manager, and click Package Manager Console.
  + In the Nuget Package Manager Console, download the latest ADF Management nuget.

Install-Package Microsoft.Azure.Management.DataFactories  
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory

* (Optional) Download latest Azure Data Factory plugin for Visual Studio 2015 (Tools -> Extensions and Updates -> Online -> Visual Studio Gallery ->Microsoft Azure Data Factory Tools for Visual Studio)
* To create DR Data Factory or tear down existing factory & define Linked Services (connection managers to data sources) for our workflow, use AzureDataFactoryFoundry program
* To create all ADF pipelines required for Dual Load Process- Execute Init, Load Process, and Archive Pipeline programs in sequence.
* Defining the schedule for the pipelines based on business requirement can be done by either: (1) modifying pipeline start and optional end times, or (2) scheduling section of activities, or (3) changing dataset availability section
* Additionally, source code provides Utility programs for ADF login, ADF output monitoring, generating ADF components, and accessing Azure resources outside of the Data Factory.

FINAL SOLUTION

Here is the link to Github Repository for the solution:

<https://github.com/manav0/HybridDR_ADF_Repository>

Above repository has complete Visual Studio solution containing

* All the code artifacts for Hybrid DR implementation using Azure and Azure Data Factory APIs
* All the table DDLs and stored procedures for Control Database.

Configuration:

* The same code can be configured for different Azure subscriptions by modifying the Subscription & Tenant Ids config fields in App.config.
* The connection strings for Control database and Azure Blob storage account can be modified in DualLoadConfigs.cs.
* Data Factory component names, sql tables, or updates to queries can also be modified centrally in DualLoadConfigs.cs.

REFERENCES

Design and implementation of DR framework using SSIS by Andy Isley, SA, DIGP 

<https://github.com/aisley/Hybrid_DR_APS_SQLDW.git>

Data Factory API 

<https://msdn.microsoft.com/en-us/library/microsoft.azure.management.datafactories.aspx>

Link for getting the ADF SDK as nugget package 

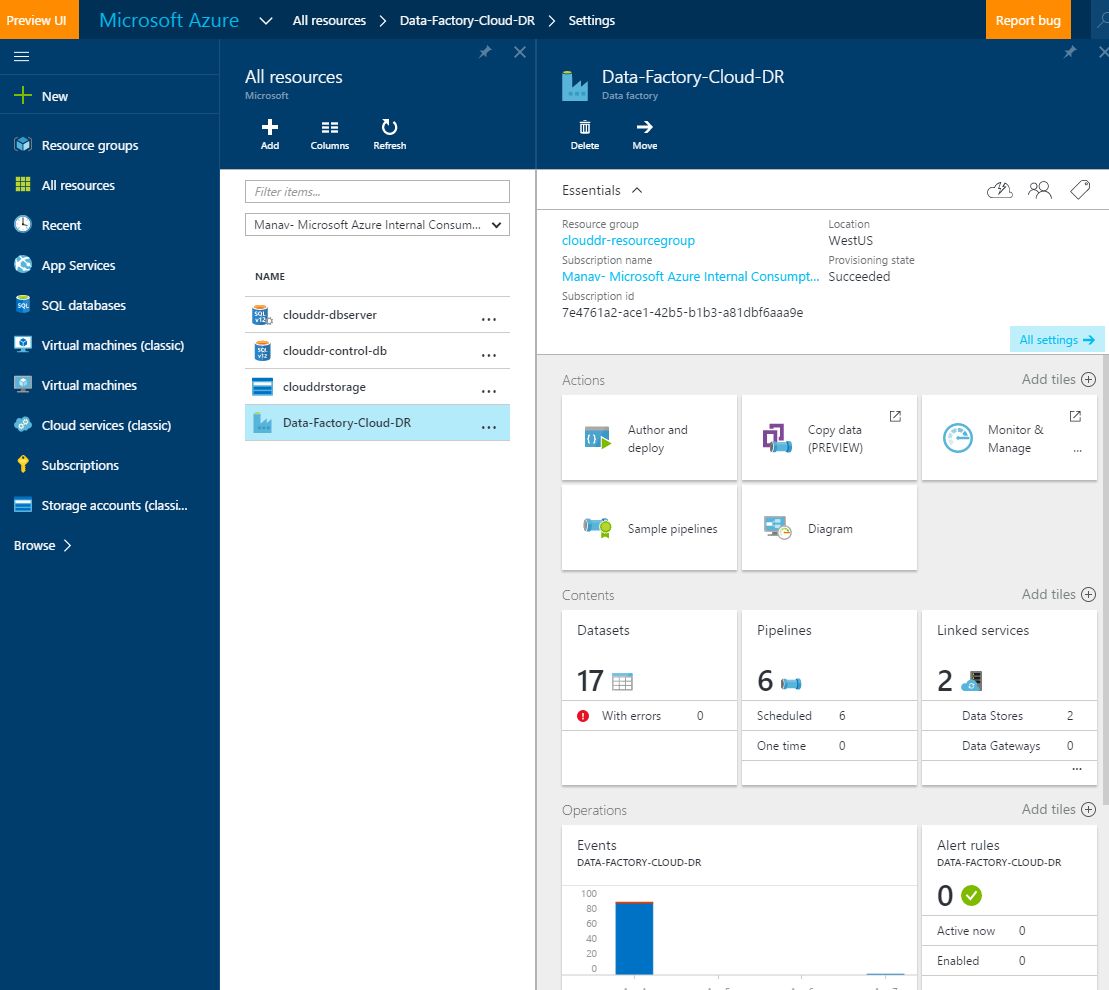
<https://www.nuget.org/packages/Microsoft.Azure.Management.DataFactories/>

Basic Example for ‘create and update’ of ADF components programmatically 

<https://azure.microsoft.com/en-us/documentation/articles/data-factory-create-data-factories-programmatically/>

Azure Portal sample snapshots for HybridDR Data Factory

Using the Azure Portal, you can browse the completed Data Factory using the diagram view to see the final result. Below Samples are based on workflows for loading 2 source data files. # of Pipelines will grow/reduce dynamically based on number of source data files

HybridDR Data Factory Dashboard:

HybridDR Data Factory (Diagram view):

