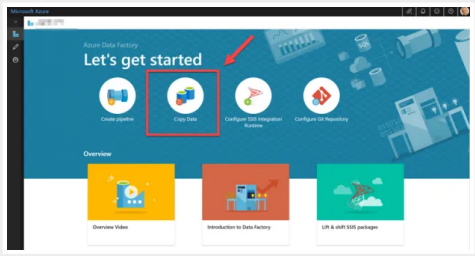
How to use Azure Data Factory V2 Sliding Windows for SQL Exports to Azure Data Lake

by Steve Young on June 3



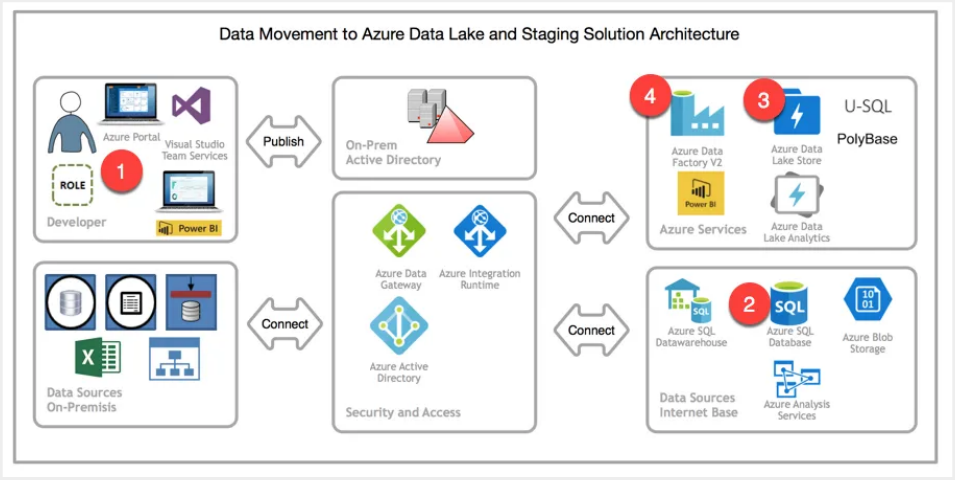
The following article reviews the process of using Azure Data Factory V2 sliding windows triggers to archive fact data from SQL Azure DB. This process will automatically export records to Azure Data Lake into CSV files over a recurring period, providing a historical archive which will be available to various routines such as Azure Machine Learning, U-SQL Data Lake Analytics or other big data style workloads.

Scenario

In the following scenario, an application is writing records to a fact table and includes a date time column, that will be used to drive a sliding window export. There is a need to get these records out of the production system and archive them for future analysis and historical requirements before the table is trimmed by another process. The team wants to create an automated process that will capture the records and copy them to Azure Data Lake Store at a predefined interval. Using Azure Data factory V2 and a trigger that runs a pipeline using the tumbling window functionality will fit the bill.

Architecture

There are a number of moving parts that make up the architecture of the overall solution. The numbered points are highlighted below and are part of a larger project.



ABOUT ME

Steve Young
Cloud Solution Architect
Data & AI
Microsoft Canada

Search

RECENT POSTS

Learning Path: Azure Synapse (Formerly Azure SQL Data Warehouse)
Learning Paths Introduction: Guided Technical Enablement
Learning Path: Azure Synapse Analytics
How to Keep up with What's New in Azure SQL Databases
Microsoft Azure Data Exams DP-200 & DP-201 Changes

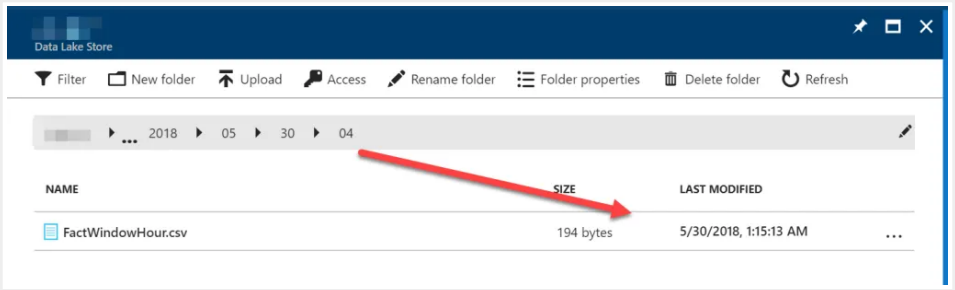
CATEGORIES

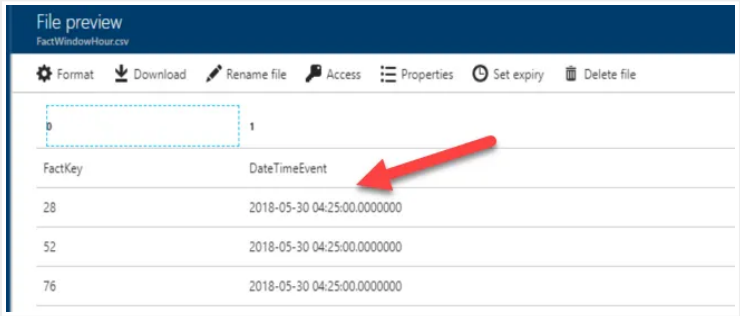
5 Minute BI (2)
Azure (7)
Azure Architecture (1)
Azure Data & AI (3)
Data Visualization (11)
Excel (6)
Hands On Labs (5)
How To (2)
How-To (5)
Learning (27)
Learning Path (1)
News (6)
Office Online (1)

1. The Developer will use the Azure Portal, Visual Studio or Power Shell to develop the objects required.
2. An SQL Azure DB acts as the source for the example.
3. Azure Data Lake Store is created with a folder to house the output. The output of the process will store the files in a separate cascading directory for each, Year, Month, Day and Hour. The process will create the folders as needed and the wild carding is included below.
4. Azure Data Factory V2 controls the process. At the time of this article the product is still in preview and may change before release.

A Short Note About Time

The goal is to have all records for each time frame, here by hour, in Azure Data Lake organized by folders. As pictured below, there is a folder in the data lake, Root/History/RawFilesByDate/Year/Month/Day/Hour/<<Filename>>.csv. The browser is in the Eastern Time Zone, but the schedule is running by Coordinated Universal Time (UTC) time. The example below shows the folder is 04 UTC, the data captured is from the table at 04 UTC and the file display time is 1 am in the browser, EST.





The window of the run will be for an hour set for UTC. If you need to make sure that the file exported, for example, only has records for the Eastern Time Zone for March 30th, you should you select the correct UTC time when setting up your window.

Most systems should use UTC time to store and then have the down steam systems translate the time required for reports later in the process. More information on time slices used in Azure Data Factory is available in the article Scheduling & Execution with Data Factory.

Source Table

For this example, the following table has been created. The key here is the **DateTimeEvent**, which will drive the window selection used by the process.

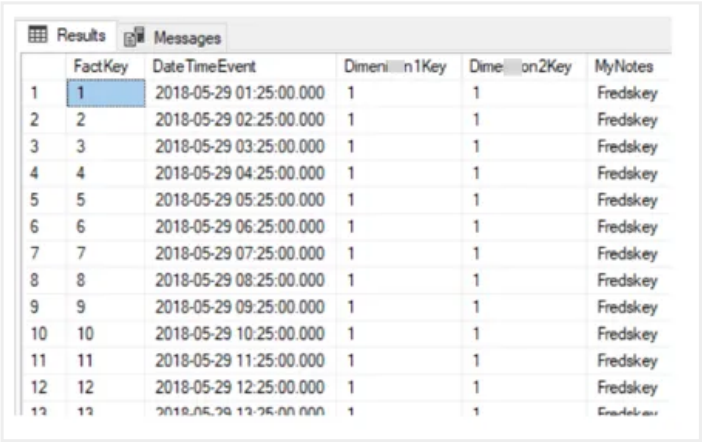
```
CREATE TABLE [dbo].[FactWindowHour](
     [FactKey] [int] IDENTITY(1,1) NOT NULL,
     [DateTimeEvent] [datetime] NOT NULL,
     [Dimension1Key] [int] NOT NULL,
     [Dimension2Key] [int] NOT NULL,
     [MyNotes] [varchar](50) NOT NULL
) ON [PRIMARY]
```

Power BI (14)
PowerBI Field Guide (6)
Quick Tips (10)
Resources (27)
Software/Services (6)
Tutorial (3)
Uncategorized (8)
Work Life Balance (7)

FOLLOW ME ON TWITTER
My Tweets

Sample data

Keeping this example really simple, one record for each hour was added to the table for a couple of days. The end data file should have one record for each export.



Results | Messages

	FactKey	DateTimeEvent	Dimen n1Key	Dime on2Key	MyNotes
1	1	2018-05-29 01:25:00.000	1	1	Fredskey
2	2	2018-05-29 02:25:00.000	1	1	Fredskey
3	3	2018-05-29 03:25:00.000	1	1	Fredskey
4	4	2018-05-29 04:25:00.000	1	1	Fredskey
5	5	2018-05-29 05:25:00.000	1	1	Fredskey
6	6	2018-05-29 06:25:00.000	1	1	Fredskey
7	7	2018-05-29 07:25:00.000	1	1	Fredskey
8	8	2018-05-29 08:25:00.000	1	1	Fredskey
9	9	2018-05-29 09:25:00.000	1	1	Fredskey
10	10	2018-05-29 10:25:00.000	1	1	Fredskey
11	11	2018-05-29 11:25:00.000	1	1	Fredskey
12	12	2018-05-29 12:25:00.000	1	1	Fredskey
13	13	2018-05-29 13:25:00.000	1	1	Fredskey

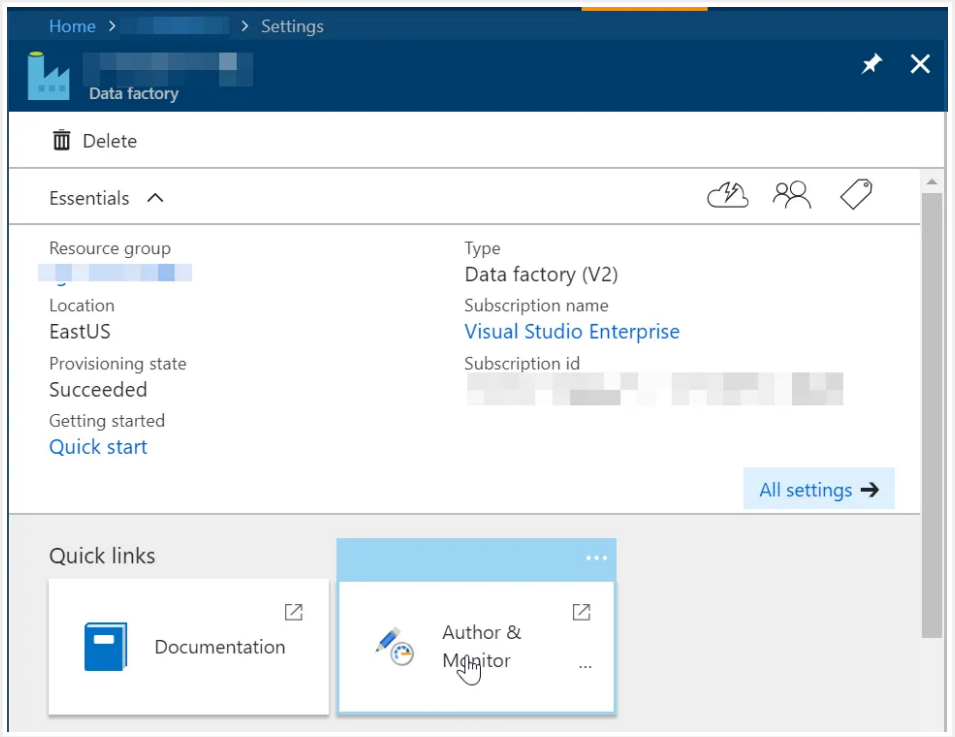
The goal is to have an export captured for each hour. The process is driven by the **StartTime** and **EndTime** of the window you create. Make sure that if you need the exact timeframe in each file, let's say an accounting application that needs all the records from the 30th , make sure that your window covers that exact timeframe.

Setting up the Process

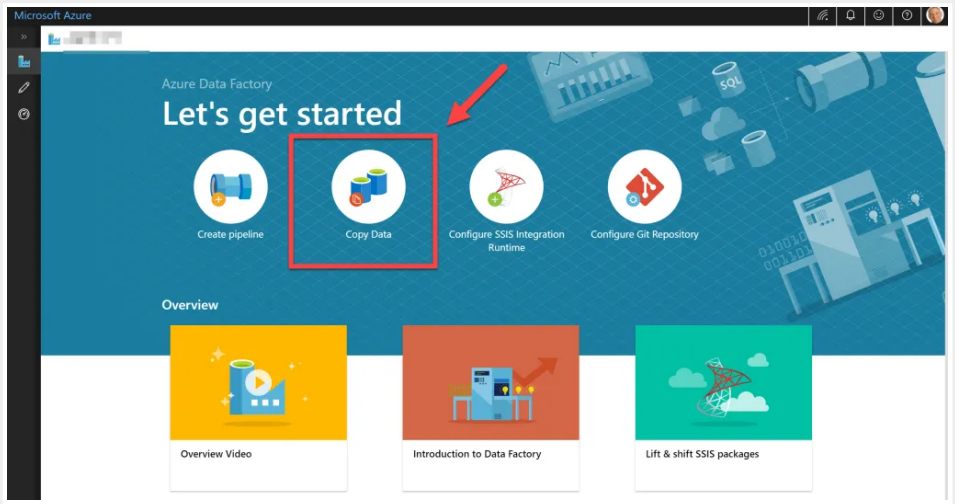
You can use PowerShell and program this out, especially if you are migrating these processes from Development to Production, however using the Wizard is by far the best way when you are just starting out. This process also assumes that you have the Connections to the SQL DB and your Azure Data Lake Store already setup.

Starting the Process

Start by going to the Data Factory blade in the Azure Portal. Select **Author & Monitor** from the **Quick Links**. The following options reviewed here will change depending on what schedule you choose, such as a one-time run or a schedule that runs every hour.



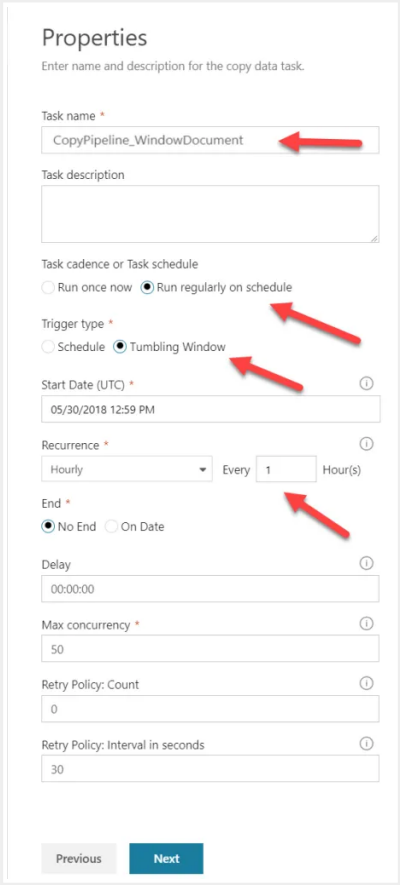
The Wizard has a number of options, for this example we are going to Copy Data.



Properties

The first blade will be the **Properties** selections for the pipeline. Make sure that you change the generic object names in the defaults that will be created during the process, this will make the editing and maintenance easier later on. Having a naming convention will allow all you to find what schedule and objects you need once you have a bunch of pipelines.

The **Properties** section will create the Trigger.



Task name: Change to something more meaningful while using a naming convention to allow for clarity later on.

Task Description: Have something meaningful here also, once you have twenty or thirty of these, commenting will save your bacon!!

Task Cadence or Task Schedule: You can run this once, or on a schedule, which we will choose here.

Trigger Type: This will create either a schedule that runs and produces the same output each run, or the **Tumbling Window** which allows us to capture activity in a specific timeframe which is this example.

Start Date (UTC): Our conversation about time earlier comes into play here. The schedule is set up by UTC time. If your data is in Eastern Time Zone (EST) and needs to change as daylight savings time comes into effect, you need to change your time here as time goes on. Try and keep to UTC for your selections, it becomes easier. For example, if you want to capture ETZ activity for a day, UTC is 4 hours ahead of EST. A good calculator can be found online. Test, Test, Test!!!

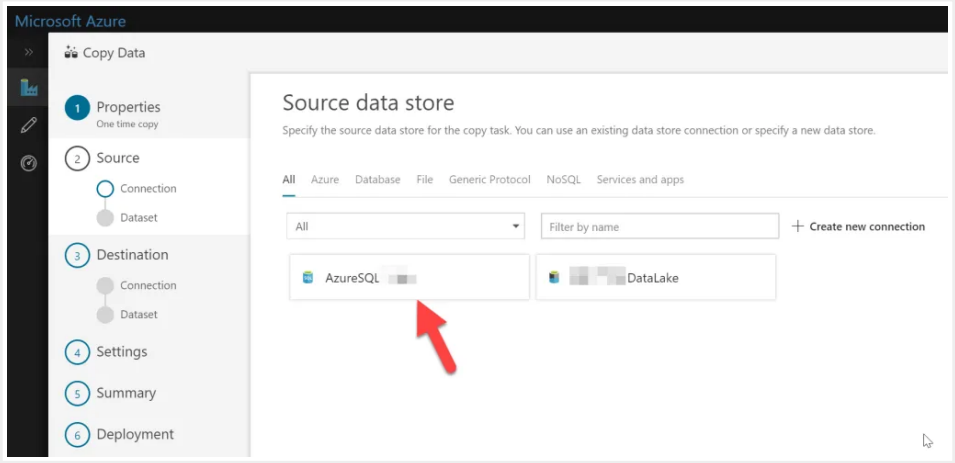
Recurrence: This sets up how often you want the schedule to run, but also what the size of the window will be.

End: This allows you to set a specific end time for the schedule, or let it run forever.

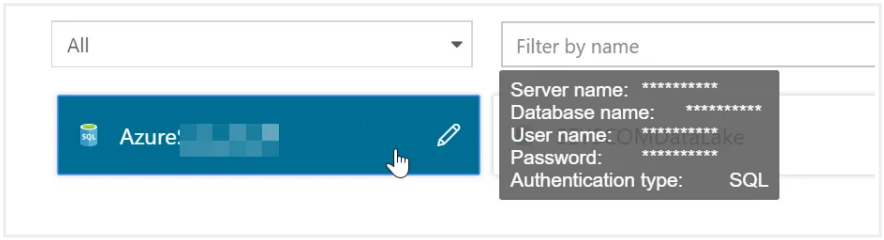
An important point is that you cannot change the start time or interval of a sliding window trigger once created. You have to create a new one. I can see the point as what happens if you change the window and miss data. Not being able to change the time is most likely because the windows are created in the background.

Select the Source

Hitting **Next** brings up the next input set, the **Source**. You can create a new connection or use one listed on the screen. For this example, we select a previously created Azure SQL DB source.



Highlighting over the sources does provide some detail, also gives you the ability to edit, however sensitive information will not be displayed.

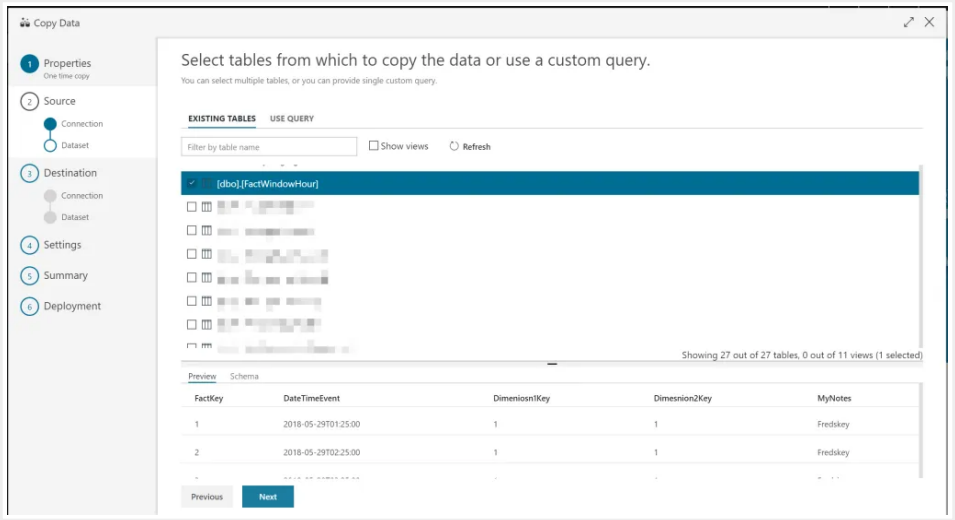


Select **Next** at the bottom of the screen.

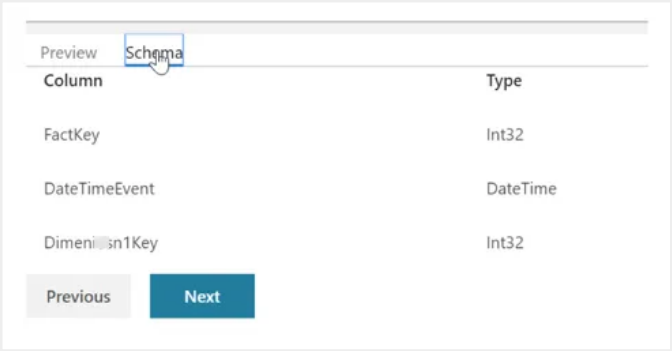
Selecting the Data Set

The next step we will select the data set we want to capture.

The capture below shows the option to select a table. The Preview will display once selected, and when selecting the Schema, you will see the discovered format of the table. This produces a select (*) so that any new columns added to the table will be included automatically. Otherwise, you have to change the query as the table changes. What you do will depend on what your requirements are downstream.



The discovered Schema will display.

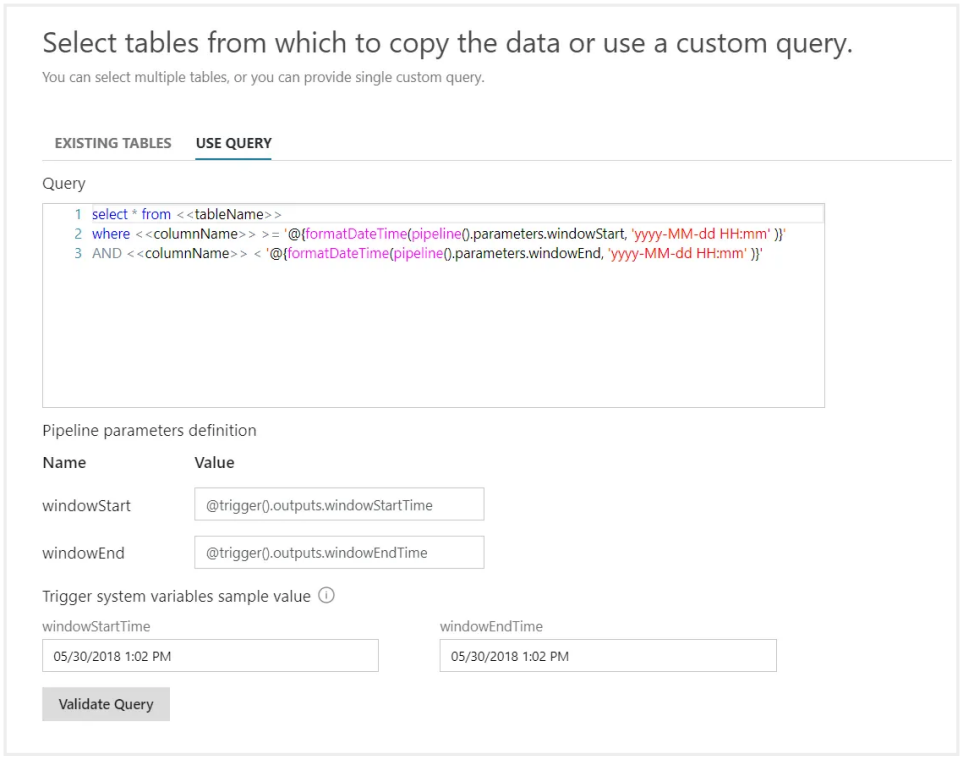


You can select this, but we want to bring in a specific query. If you do this format, you will have to change the query if any more columns are added. The default is "Select *", however we will paste in a query in this example. A query could join a number of tables, but for ease of the example, we will only select the one table.

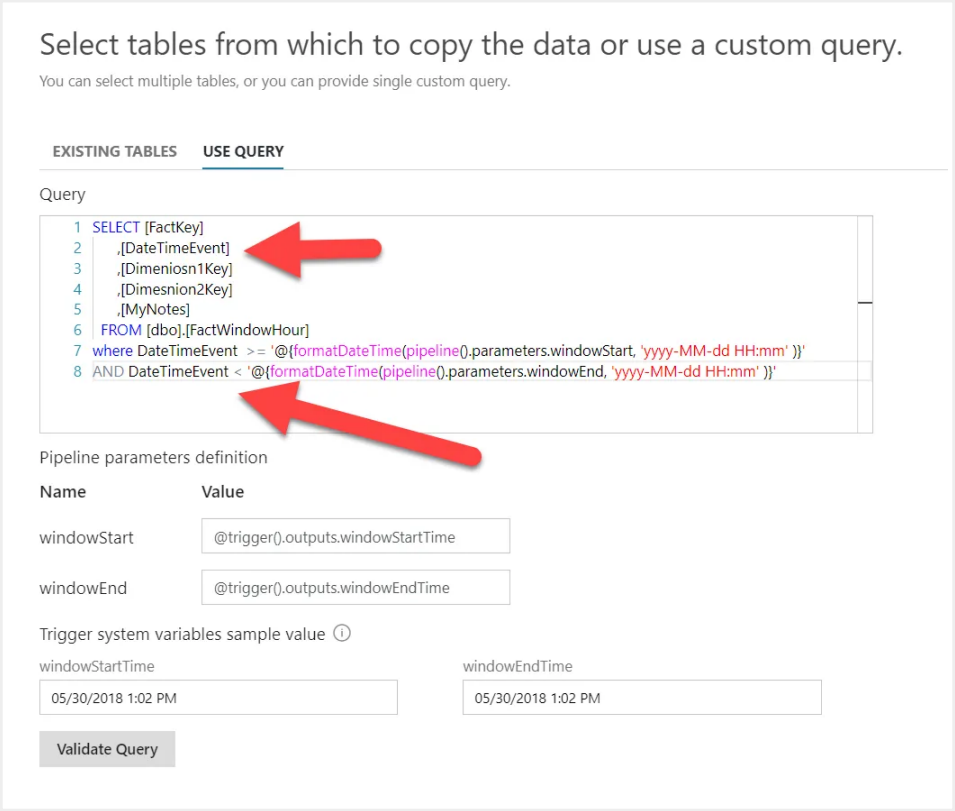
We selected the tumbling window, so the following meta data has been added.

Edit the Query Window

This screen allows us to have a couple of options for our query.



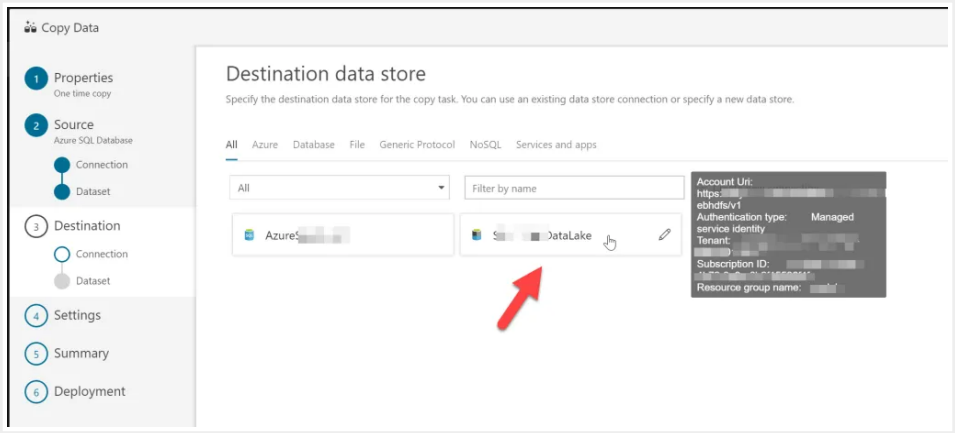
We paste in the select query code and hit the Validate button. In the **<<columnName>>** placeholder, put the date time column you will be using to drive the window. The below capture shows the validated query with the discovered Schema.



Now hit **Next**.

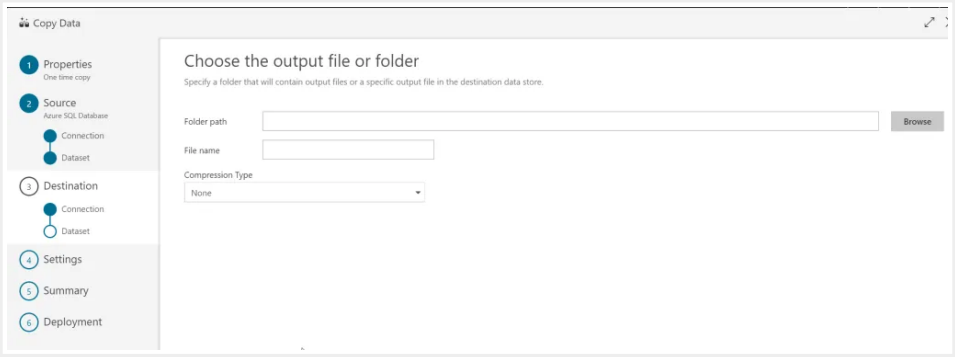
Select the Destination

Now select the Destination and hit **Next**. Highlighting the icon will show the details and also provides you the ability to edit the Destination Connection if required.



Dynamic Folder Paths

Selecting **Next** allows you to select the Folder Path

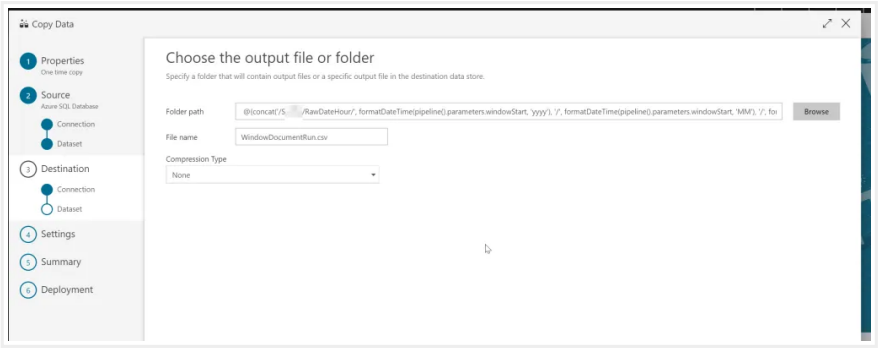


You can browse to the folder you want to use to store the files which gives you a static option for the location, but in this example, we will provide a dynamic folder location as the default is to overwrite the file. The code below will create the cascading folders for each run. This version goes to Hour and will have the file name being the same for each run. Use the same format for File Name if you want to go to that level.

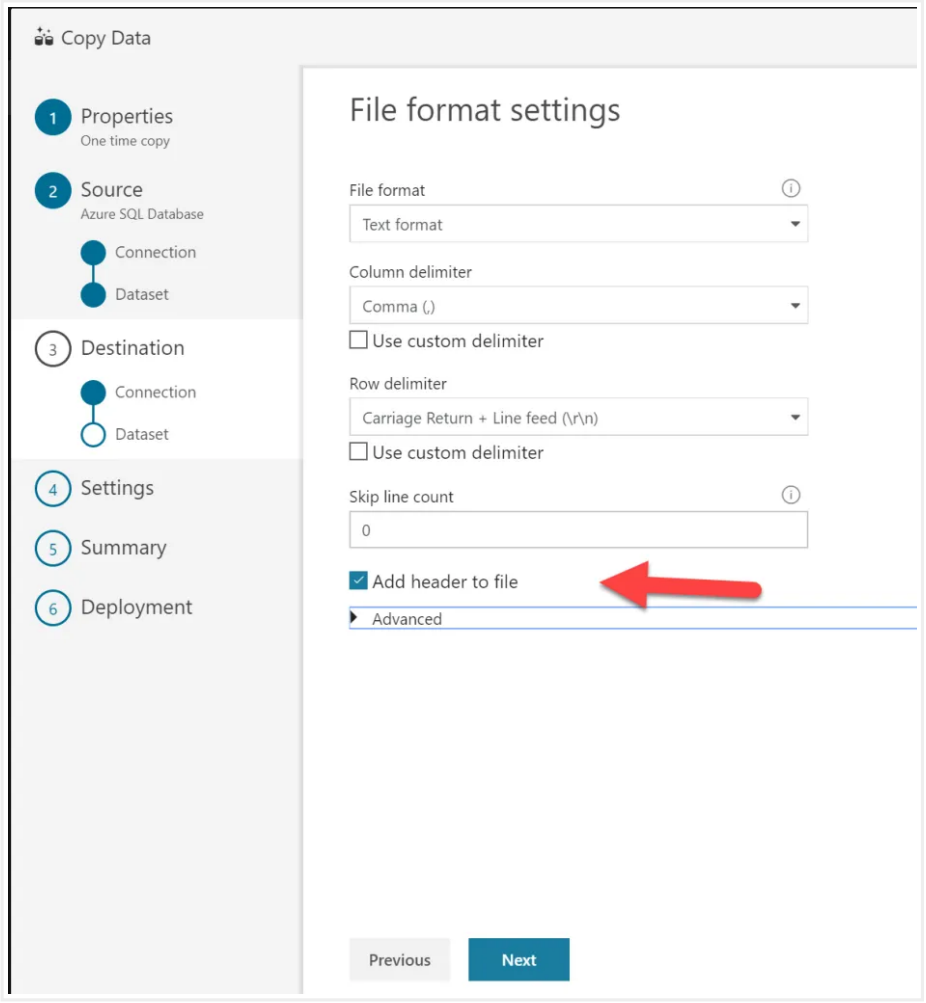
```
@{concat('/SystemRun/RawDateHour/',
    formatDateTime(pipeline().parameters.windowStart, 'yyyy'), '/',
    formatDateTime(pipeline().parameters.windowStart, 'MM'), '/',
    formatDateTime(pipeline().parameters.windowStart, 'dd'), '/',
    formatDateTime(pipeline().parameters.windowStart, 'HH'))}
```

Note: You can also use "**Partition by**", when writing to Azure Blob Storage. See Example.

It is very important to note that the capital "HH" provides the 24 hour clock hour, and the capital "MM" provides a zero padded month, "05" for example.

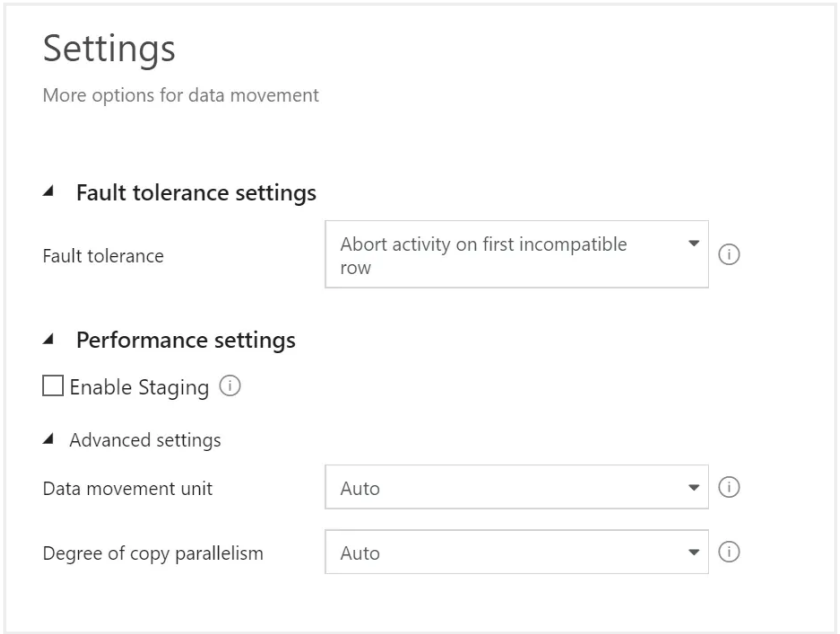


The next screen, provides settings on the file. I usually add in headers, but not always, it will depend on you the requirements of the down steam workflows. In U-SQL for example, you ignore the headers on import. Just be consistent by project !!!!!

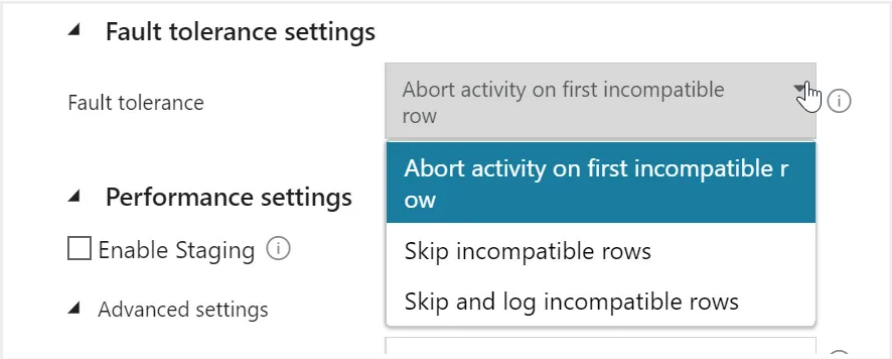


Setting up Fault Settings if Things go Wrong

On the settings tab, you can set what happens to the package if there are problems in the rows. Normally, Skip and Log Incompatible rows may be the best way to handle this, but you may have something specific you need to do in the flow once an error occurs.



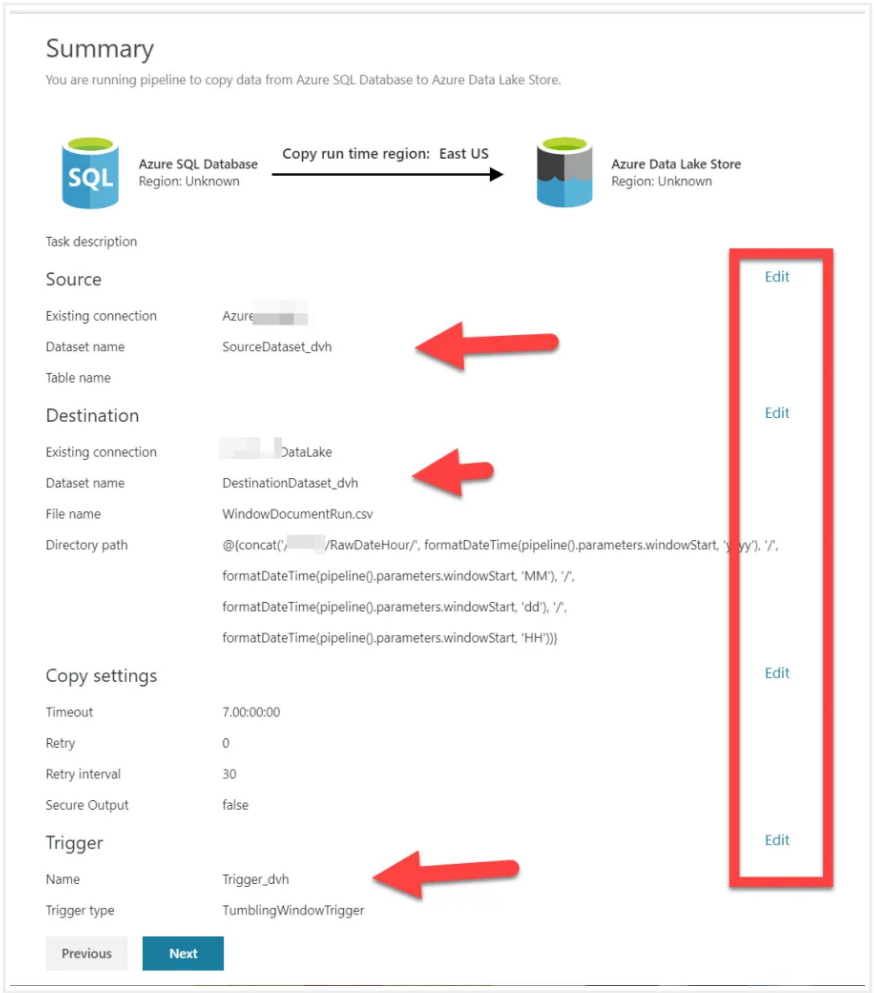
Various Settings for faults are listed below



Note: See Rerunning Failed Data Slices and Automatic Data Dependency Tracking, which includes how to monitor execution of slices visually and how to re-run failed slices.

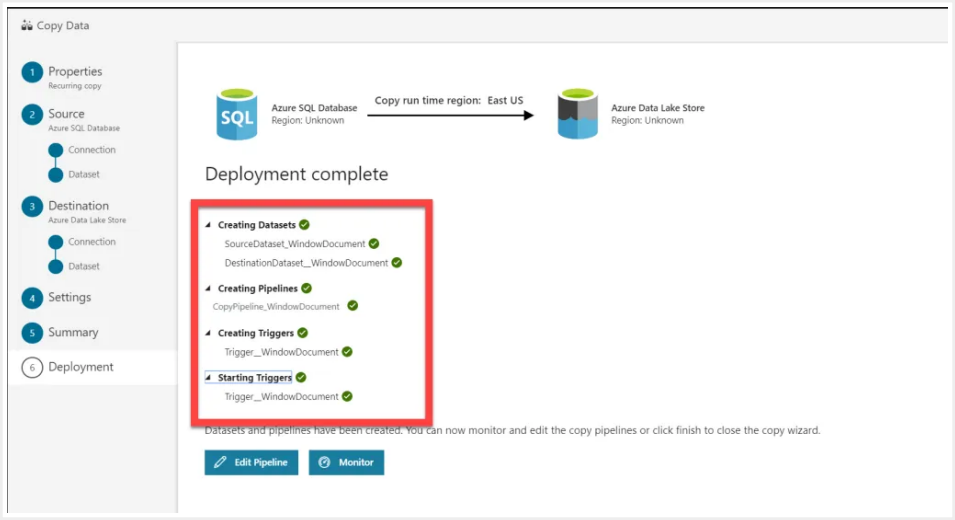
Summary

Now hit **Next** to show your creation summary. Note the names of the data sets and destination items. Select **Edit** to give them names that will be more descriptive later on for anything you missed during the creation process.



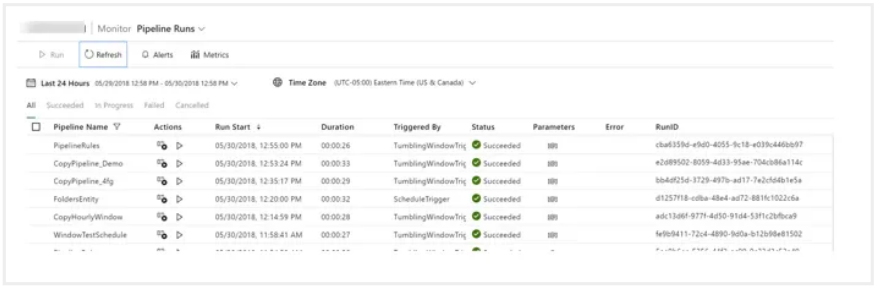
Deployment

Hitting **Next** will deploy the objects and provide the success or failure of each object.

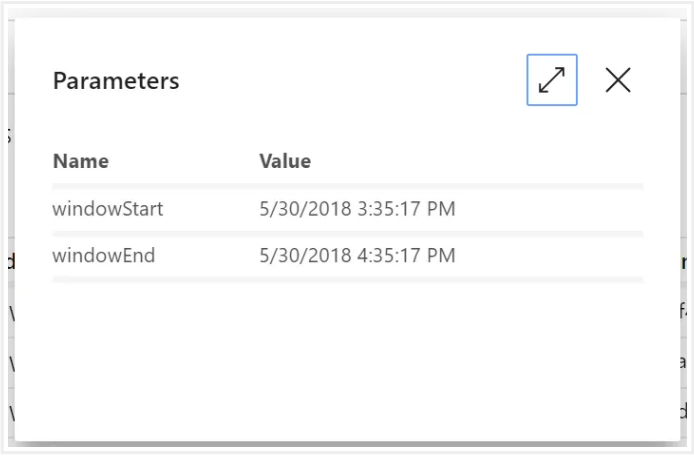


After Your First Run !!

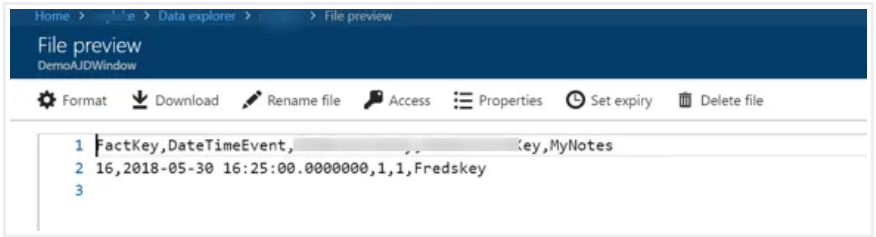
On the **Monitor** tab, you will see the runs based on your schedule and if they were successful.



In the **Parameters** column, selecting one, will show you what was used in that run.



Looking at the file, you will see the records pulled during that window.



Conclusion

Having the ability to easily set up an archive routine and a sliding window to copy data from source systems to Azure Data Lake is a real time saver. The dynamic file names and folder structure allows a data developer to easily create these objects and get them automated.

Resources

Azure Integration Runtime: The Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory to provide the following data integration capabilities across different network environments, Data Movement, Activity Dispatch and SSIS Package Execution.

Power BI Data Gateway: With the on-premises gateways, you can keep your data fresh by connecting to your on-premises data sources without the need to move the data.

Pipeline Execution Trigger:

How to create a Schedule Trigger:

How to Create a Tumbling Window Trigger

How to Create an Event Trigger

Azure Data Factory V1 Scheduling

Azure Data Factory Scheduling and Execution

Copy data to or from a file system by using Azure Data Factory

Related

Using Azure Data Lake with Power BI
September 28
In "Hands On Labs"

Learning Path: Azure Synapse (Formerly Azure SQL Data Warehouse)
December 8
In "Learning"

Learning Path: Azure Synapse Analytics
December 4
In "Azure Data & AI"

Edit this

TAGS: AZURE DATA FACTORY V2, AZURE DATA LAKE, DATA MIGRATION, SLIDING WINDOW ELT

FROM: AZURE, HANDS ON LABS, HOW-TO

← How to use Data Driven Settings to Control Power BI Reports

SSDT for Visual Studio 2017 – Fixing the Install Error →

No comments yet

Leave a Reply

Comment

You may use basic HTML in your comments. Your email address will not be published.

Subscribe to this comment feed via RSS

Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit Comment

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

Tags

Azure Automation Runbooks Azure Data lake Azure Mobile Services Azure SQL Data Warehouse Azure Synapse Analytics BI BI Examples Blogs Books Dashboards Datasets Data Visualization Data Visualization Tips ETL Excel Sample Excel Sample Workbook Free Data Sources How To Learning Resources MAC News OpenSource Power BI PowerBI PowerBI Example Power BI Example PowerBI Sample Power BI Training PowerShell PowerView Productivity Quick Tips R Resources RStudio Samples Scorecards ScreenCast SQL Azure SSIS Text Editor Tutorial Video Web App Work Life Balance

***Disclaimer**: I work for Microsoft, however, this is not an official post and the views expressed are my own. I offer no guarantees for the accuracy of the information shared and is for educational purposes only.*

All nonoriginal photography is sourced and licensed from my account on either, PEXELS or STORYBLOCKS. Please DM me on Twitter if you have a question.

Categories

- 5 Minute BI
- Azure
- Azure Architecture
- Azure Data & AI
- Data Visualization
- Excel
- Hands On Labs
- How To
- How-To
- Learning
- Learning Path
- News
- Office Online
- Power BI
- PowerBI Field Guide
- Quick Tips
- Resources
- Software/Services
- Tutorial
- Uncategorized
- Work Life Balance

Meta

- Site Admin
- Log out
- Entries feed
- Comments feed
- WordPress.org