You will need:

* A local integration runtime configured and synched with Data Factory
* A CSV with some data – SalesOrderHeader.csv is a nice easy one to start, provided with this lab

This lab guides you in building a small file checking utility – it uses several of the common General Activities to check a fie exists, return some metadata, delete the file and log an error when revalidating it. Logically, not the most useful, but a good change to try out some of those activities.



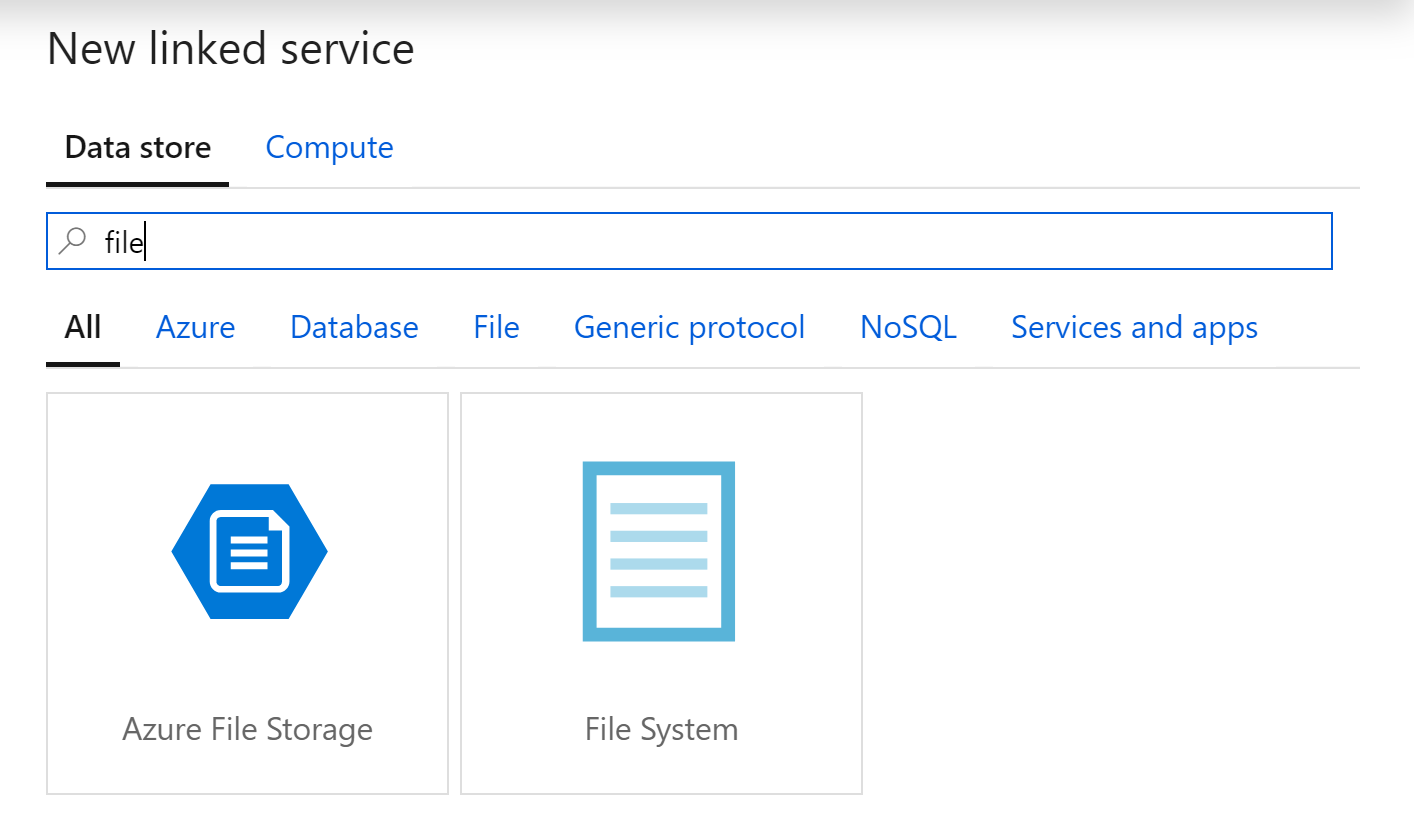
## Create our “network drive”

1. Firstly, we need somewhere to access the data. Take the SalesOrderHeader.csv and drop it into a folder called C:\Data\SalesOrderHeader (or a similar folder of your choice)

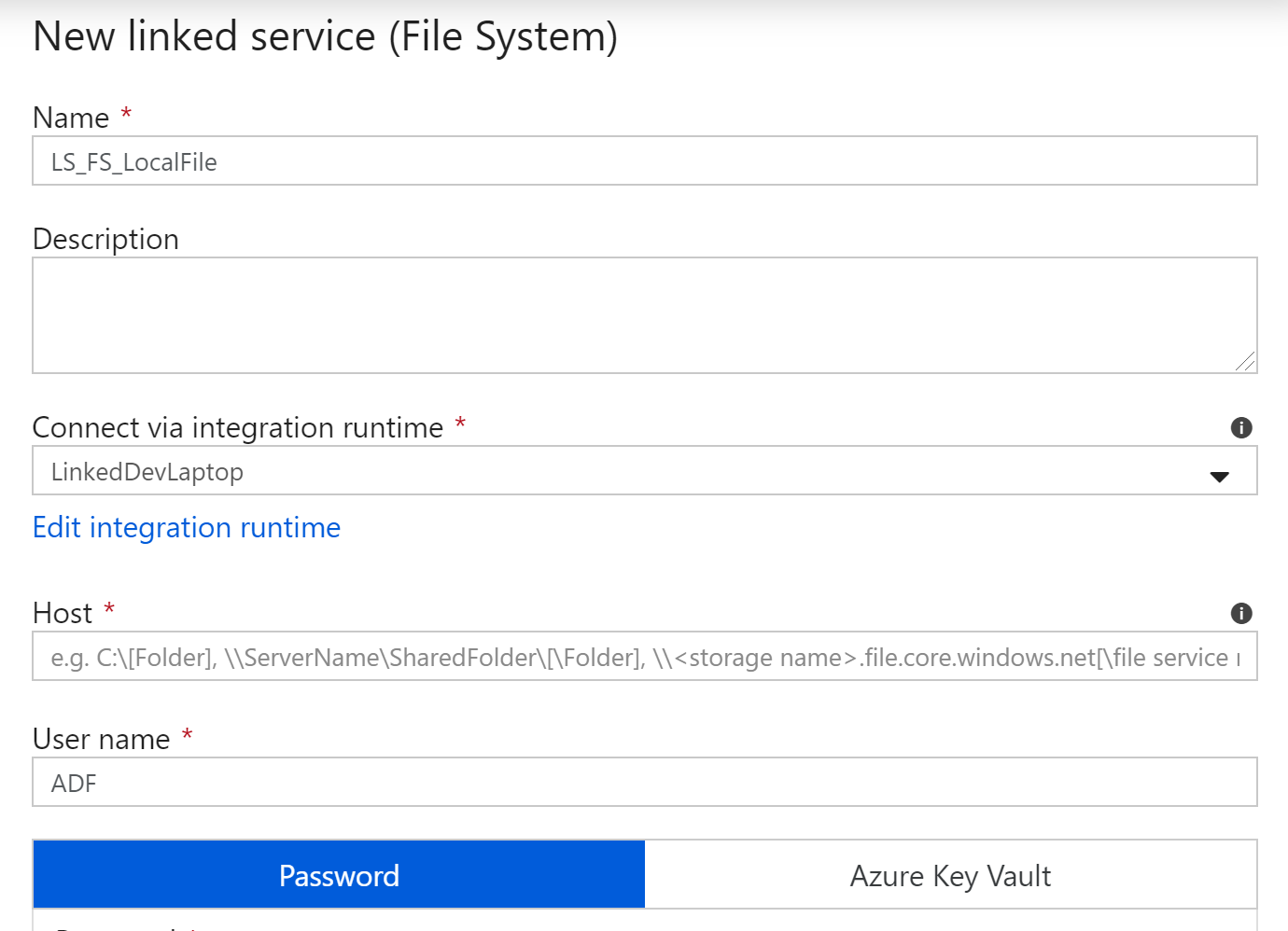
## Configure a File System Linked Service

In the previous example, we created a linked service to a SQL Server that our local integration runtime could access. In this lab, we’re going to use a local file instead.

1. Create a new Linked Service of type “File System” – make sure you use the normal file system (ie: a network drive/local folder access via self-hosted IR) not the Azure File System



1. Click continue then give the new linked service a sensible name.
2. Select the name of your self-hosted integration runtime from the drop-down list, then enter a username and password for a user that has access to the network drive (this can be a local user on your machine for this example)
3. Populate the “Host” as the folder where we dropped the sample file (C:\Data\SalesOrderHeader)

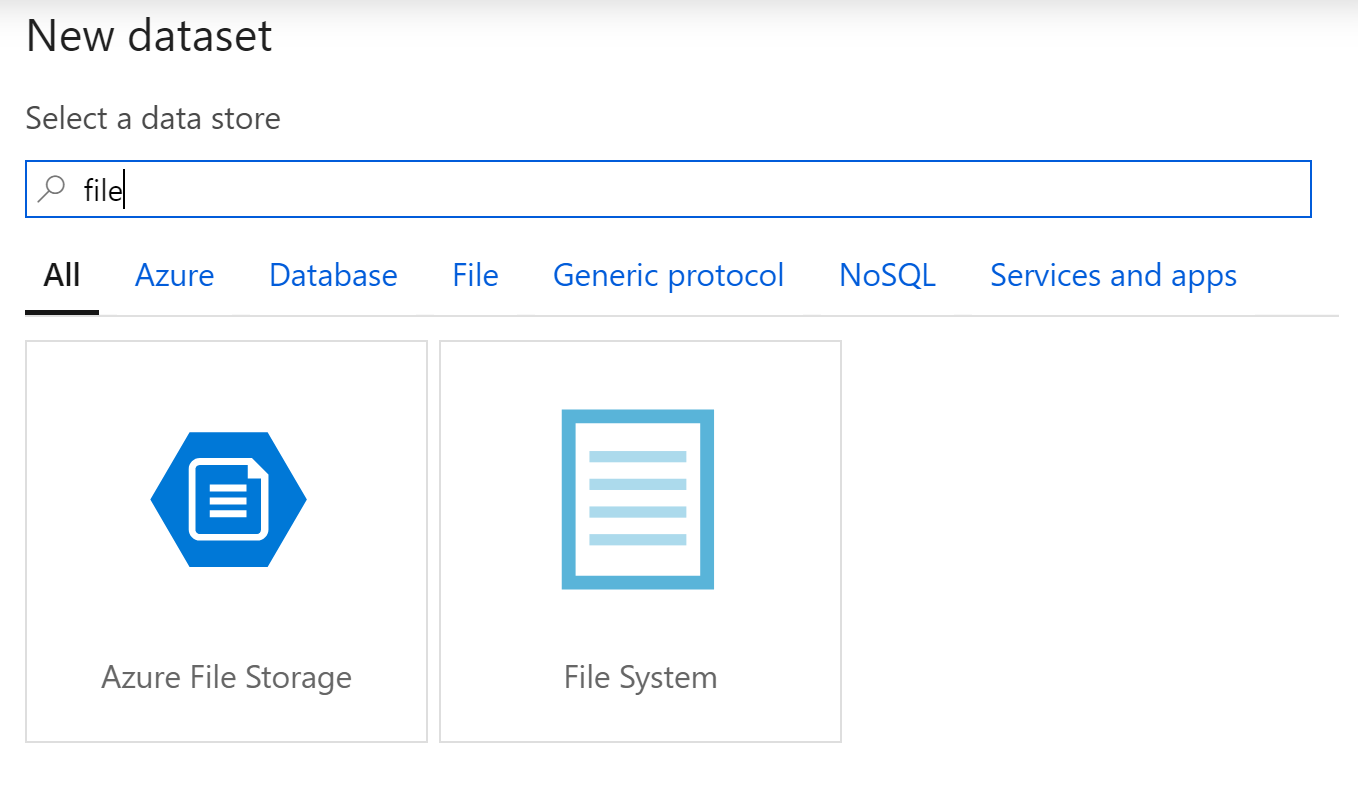


1. Test the connection and click “Create” to save your linked service

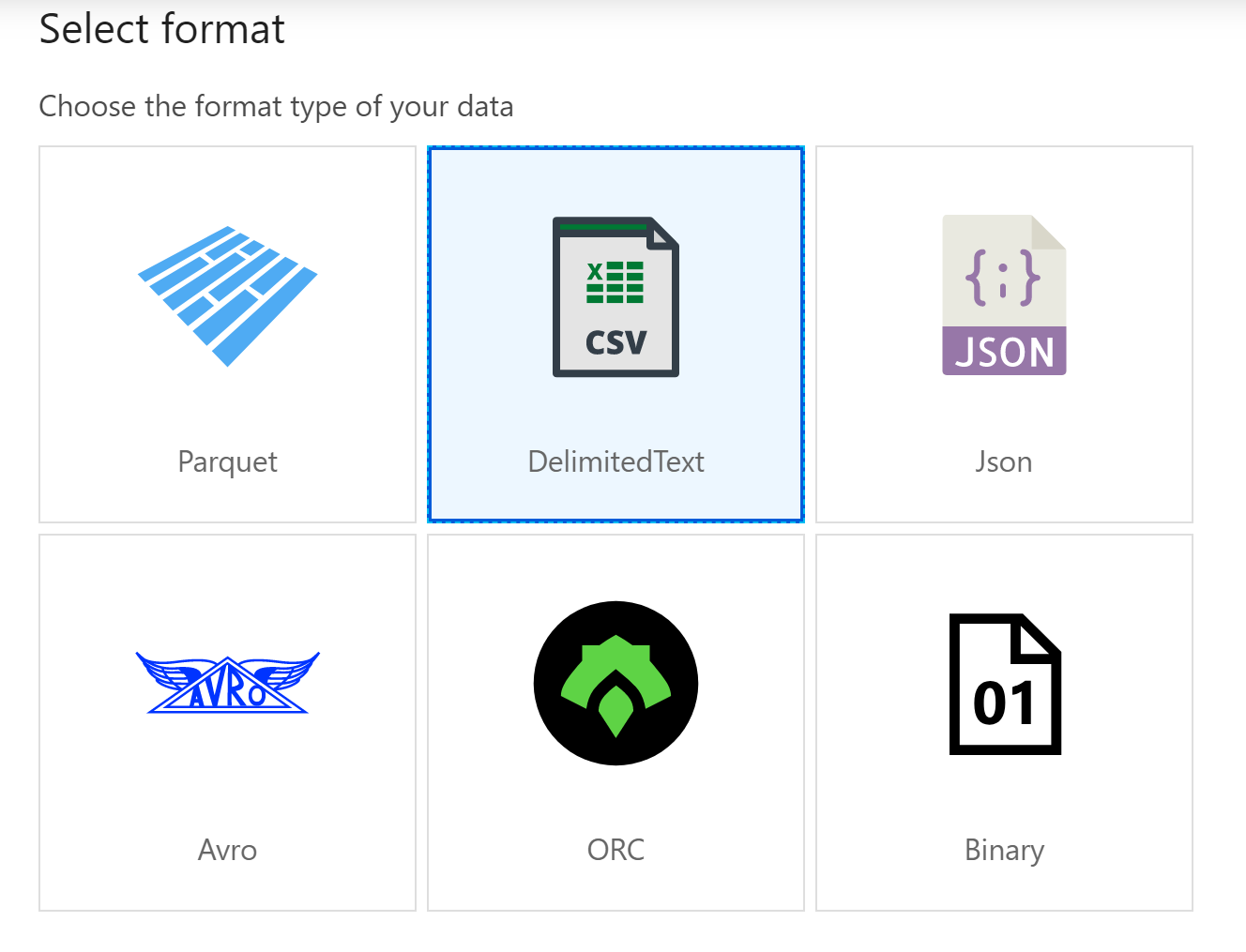
## Create a Dataset

Now we need to set up the dataset we’ll be using:

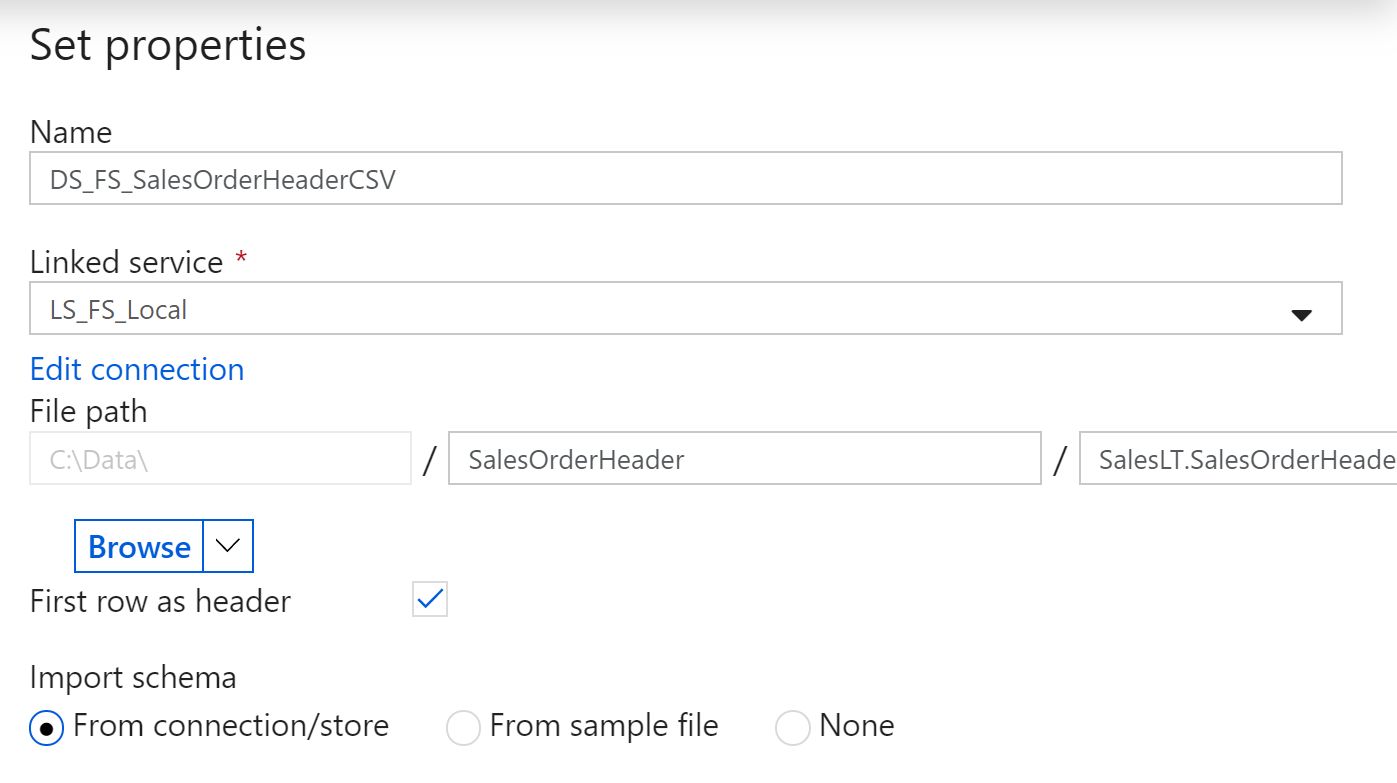
1. Click on the “Create a Dataset” button and select a dataset of type “FileSystem”, as with the linked service:



1. Selected “DelimitedText” for our CSV file



1. Name the Dataset and pick your Self Hosted Integration Runtime



1. You can then use the “Browse” button to select our CSV file
2. Click OK to create the Dataset

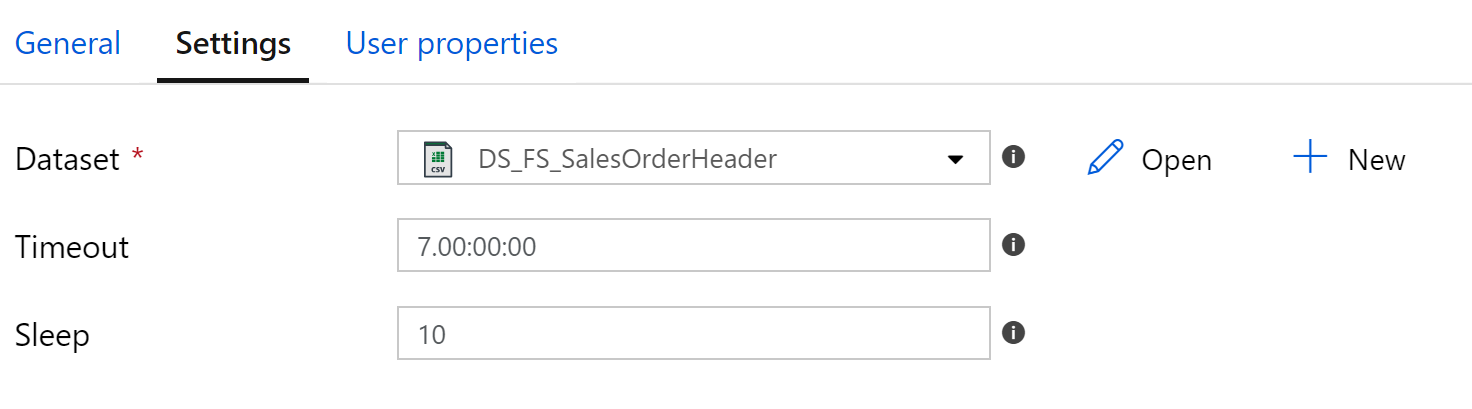
## Building the Pipeline

Now we have our connection, we can do everything in our list.

1. Create a new pipeline and give it a good name
2. First, let’s validate our file exists. Drag a “Validation” activity from the general activities drop down, onto your pipeline workspace.

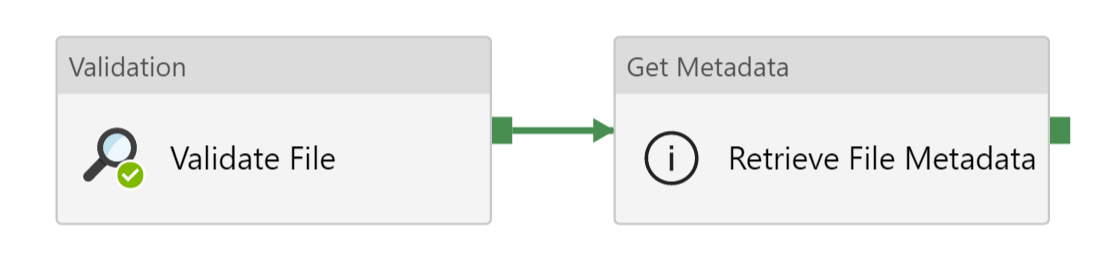
The validation activity will check if a given dataset and, if not, wait for a given time, checking the dataset periodically. Using this in a workflow allows you to stop the process until a dataset has arrived – this is great if you have various decoupled processes, but dependencies between them.

Associate the Validation activity with our CSV Dataset, and it’s good to go



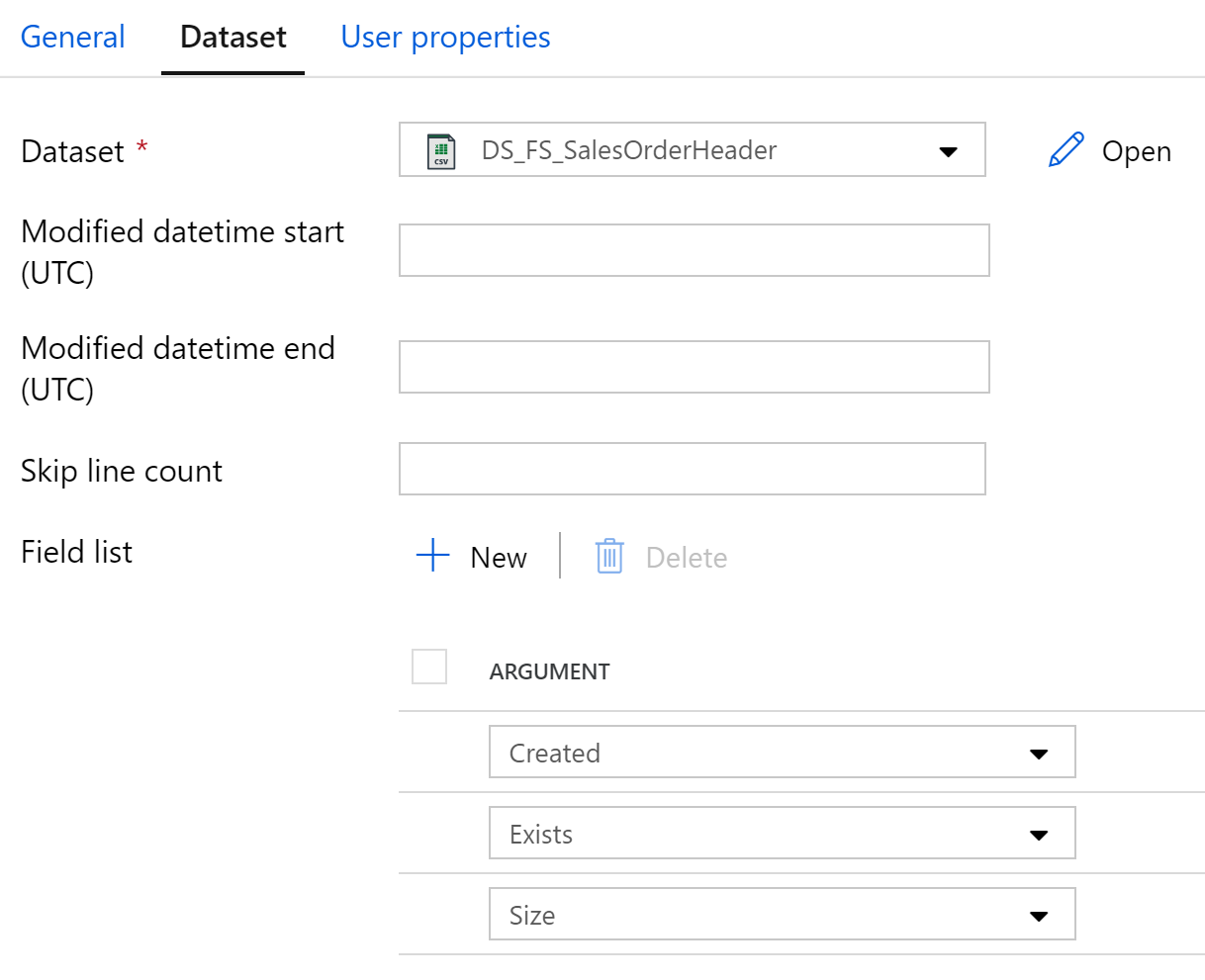
1. Now that we know the file exists, we can get some metadata about the file – this can be very useful in enriching our logging/metadata, but can also help us with some more advanced control flow further down the line

Add a “Metadata” activity to the pipeline, and drag the green connector from our Validation activity to the Metadata activity



Just like SSIS precedence constraints, this will associate the two activities and execute our metadata activity only once our validation is complete

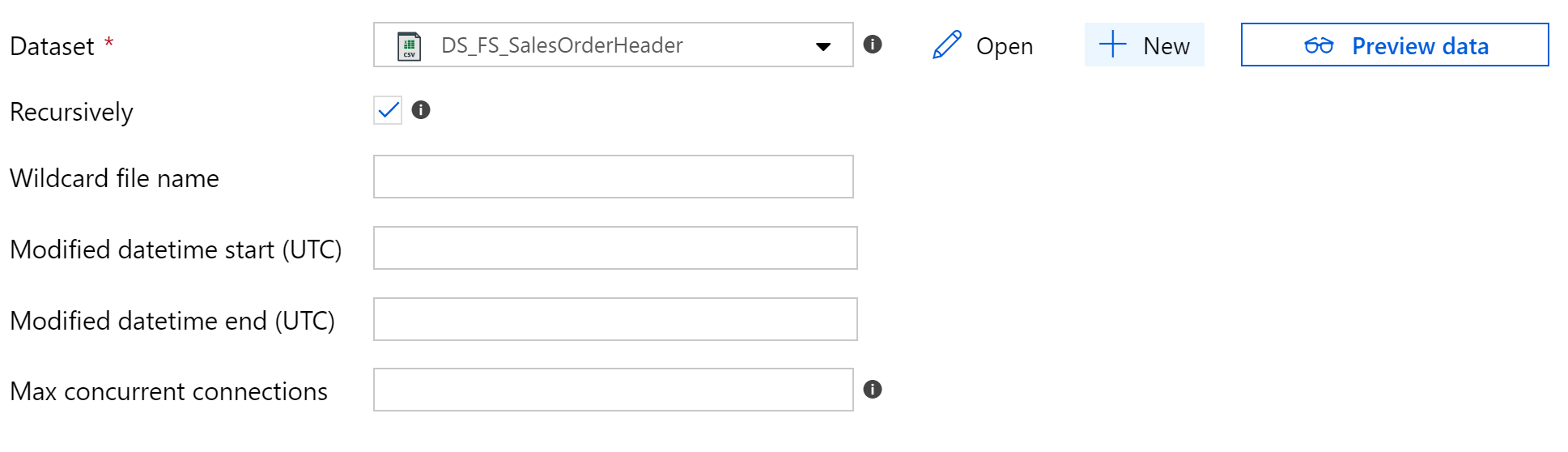
1. Configure the Metadata Activity to use our dataset, and add in several file attributes we want to return



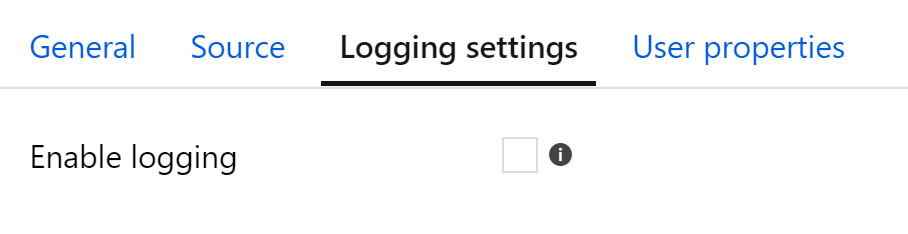
1. Now that we’ve inspected the file, let’s clean it up. We can use the “Delete” activity to remove files from file storage, drag this onto our pipeline and associate it as with the others.



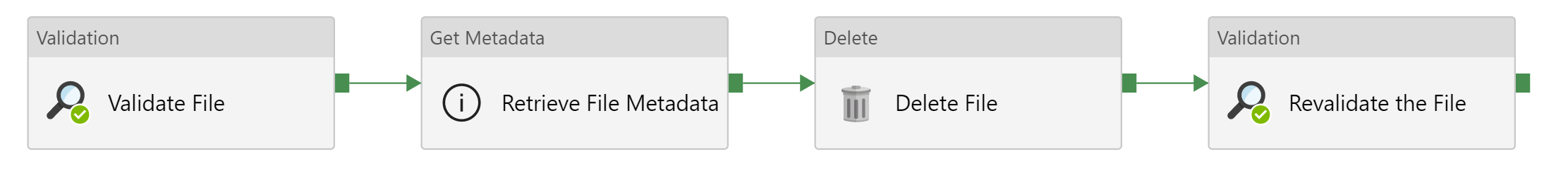
1. Configure the Delete activity to use our dataset



1. We will then need to configure logging – by default the delete activity wants to log out details of all deletions, for this example, just disable logging



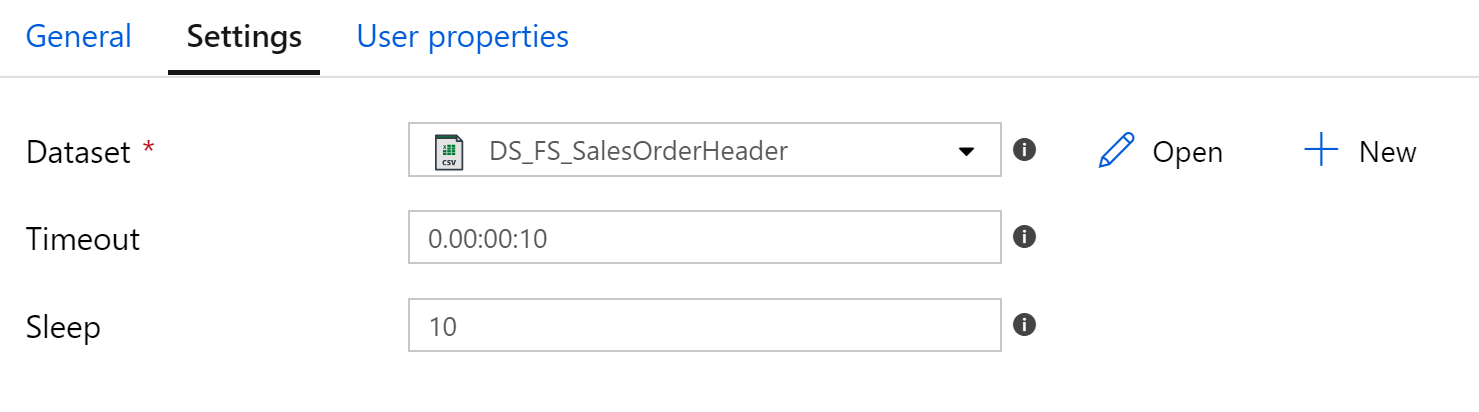
1. So now the file will have been deleted, and we can test what happens when we fail to validate. Add a new validation activity and add it to the chain:



We want to validate the same file, so associate the activity with the same dataset as before.

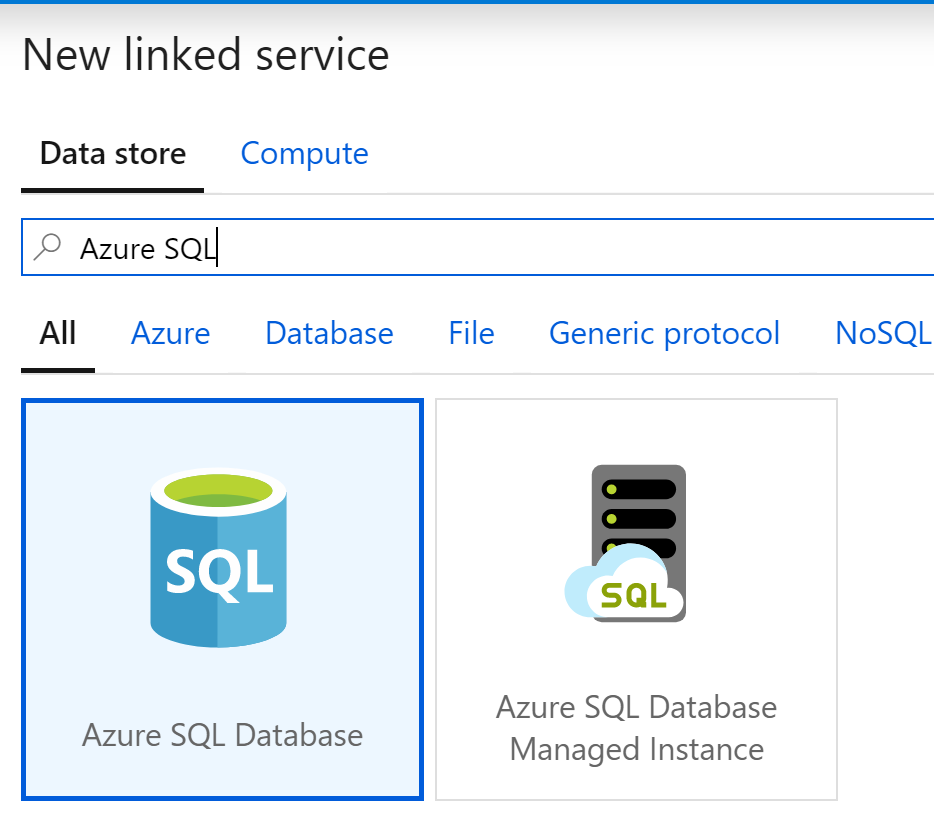
1. By default, Validation activities are VERY lenient, they will wait for 7 days before timing out (and you’ll be charged for this duration).

Set the timeout to 10 seconds, so we will definitely see a timeout failure when the activity fails to see the file. The format for this file is D.HH:MM:SS



1. Now that we have a failure likely to happen, we can set up some basic logging. We’ve provided a logging service for you, but you’ll need to set it up.

Create a Linked Service for an Azure SQLDB:



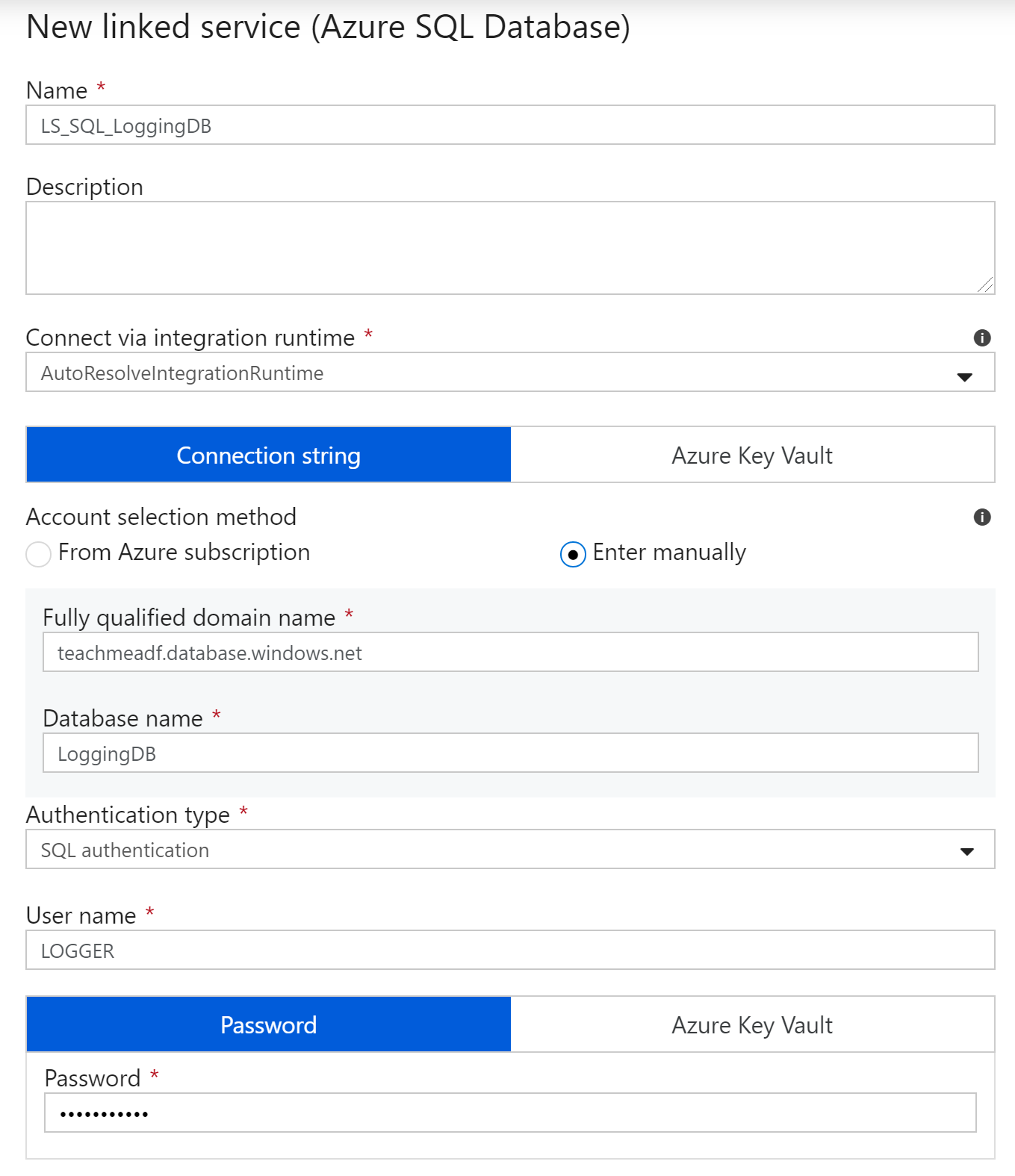
The connection details are as follows:

Servername: teachmeadf.database.windows.net

Database Name: LoggingDB

User: LOGGER

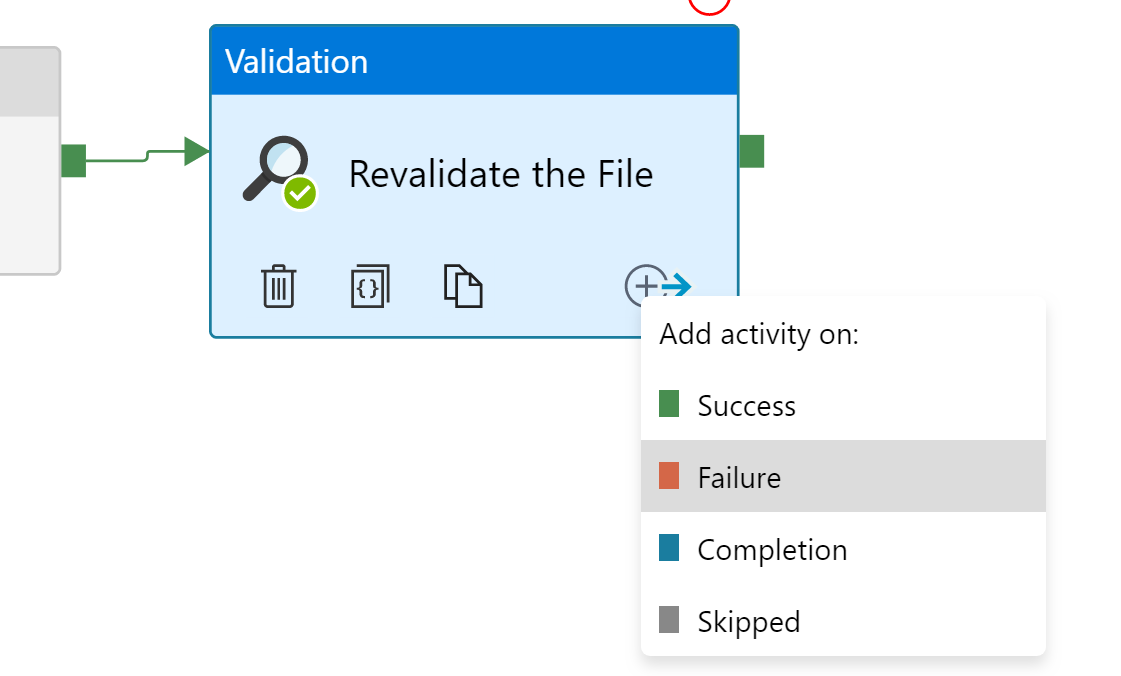
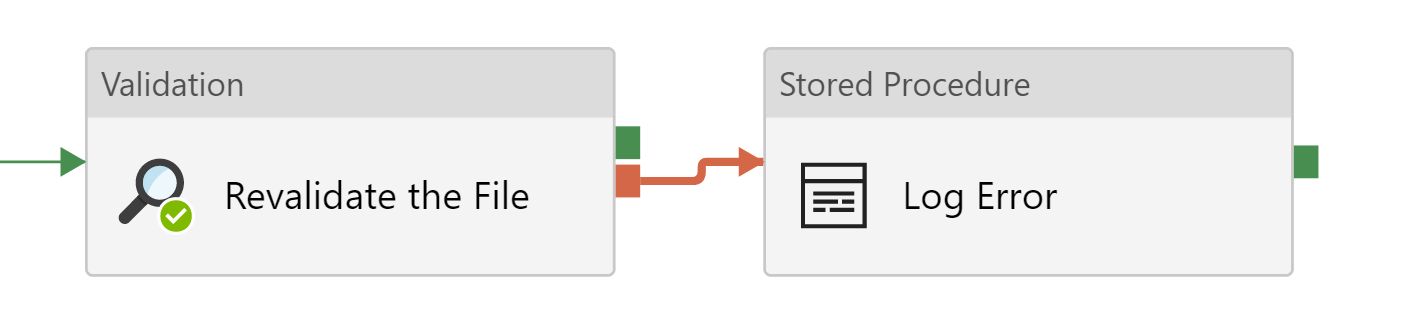
Password: Te@chme@df!



1. We’re then going to add a Stored Procedure activity to our chain – this doesn’t need a Dataset so we can skip that part!

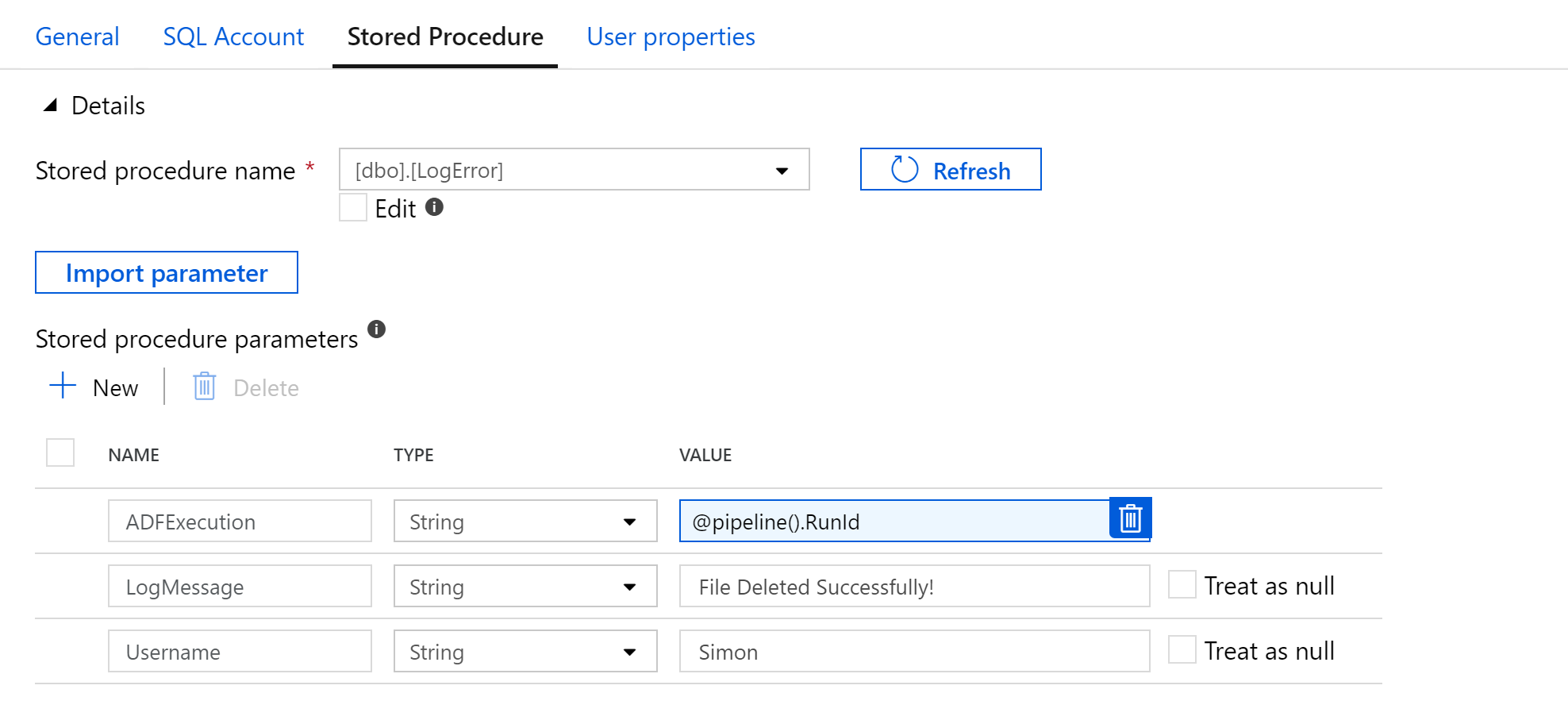
Add a Stored Procedure Activity to the Pipeline Workspace – this is going to be a little different as we will be using a different type of precedence constraint.

1. Highlight the final Validation step and click  to add an advanced precedence – in this case Failure, then join the red path to our Log Error

1. Set up the Stored Procedure to use the new Linked Service and, under “Stored Procedure”, set it to use the “dbo.LogError” stored proc.

The “Import Parameters” button will bring in the required inputs, and we can fill them manually in this screen.

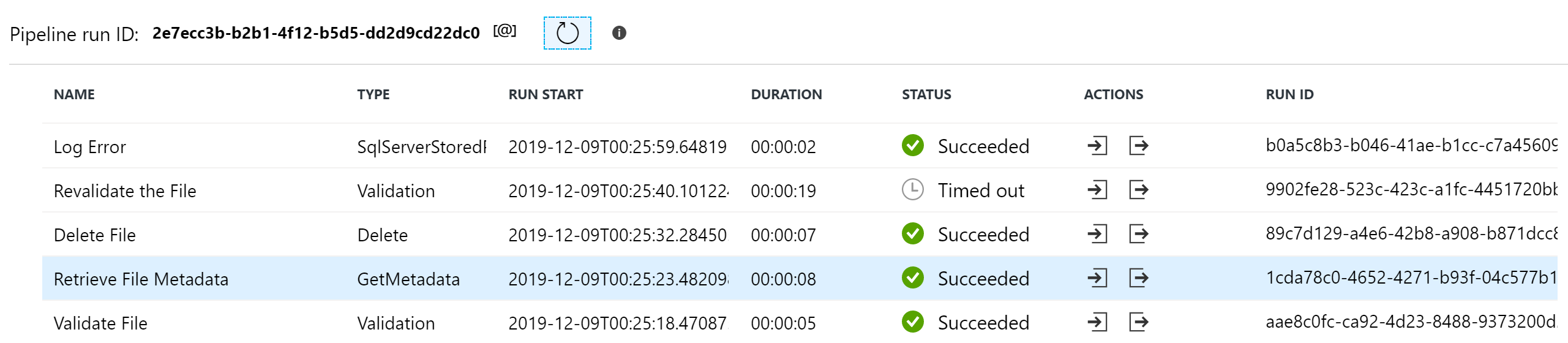


You’ll notice the ADFExecution parameter has some special code in it – type in @pipeline().RunId for now, and we’ll take a look at ADF expressions in more detail later!

## Run the Pipeline

Ok – your pipeline is built and configured – we need to test it.

Click on the  button to spin up a debugging version of your pipeline and execute your chain. If all goes well, you will see each activity execute and kick off the next, with the final stored procedure executing successfully:



Debugging your pipelines as you build them up is a vital skill to get used to!