Lab 03.1 – Implementing ForEach and Dynamic

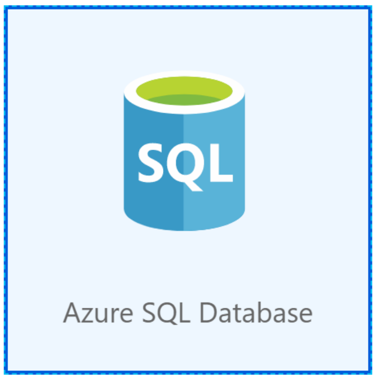
This Lab builds upon the dynamic pipeline we created in Lab 3. By creating a pipeline that can accept parameters and perform different activities based on the input.

Our next step is to automatically run this pipeline for all tables on the database and make sure it picks up new tables without any configuration!

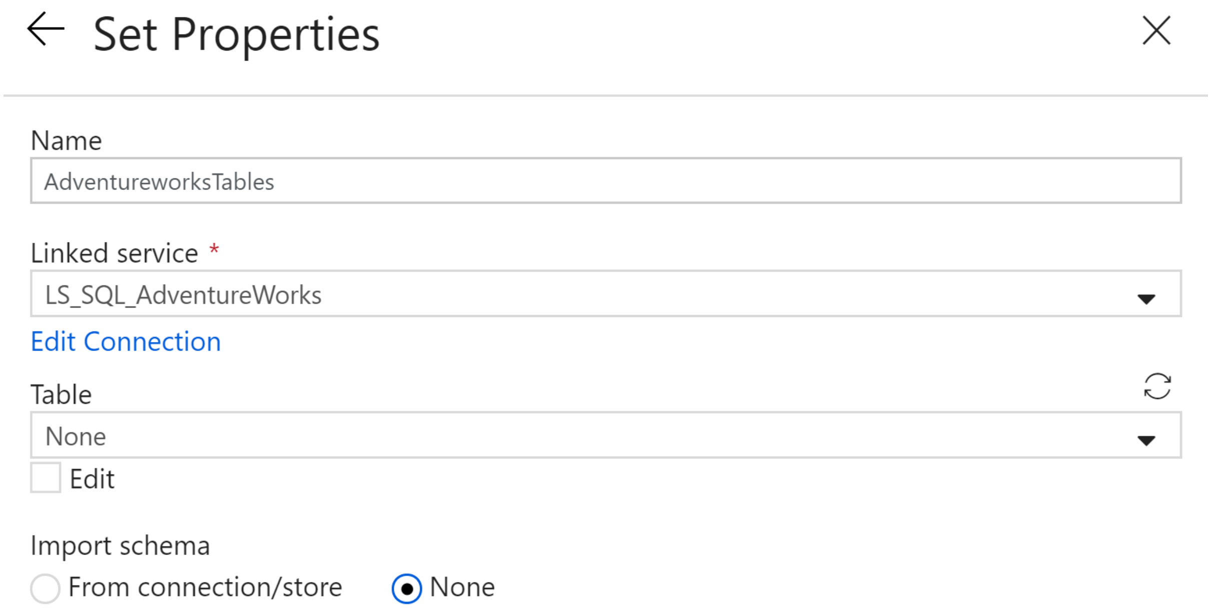
Lab 03.1.A – Create Tables Dataset

Firstly, we need something that tells us what tables exist on the source database. We've already got the linked service connecting to the Adventureworks SQLDB, but we need to create a new dataset to return a list of tables.

1. Click to add a new dataset and select Azure SQL DB



2. Associate the Dataset with the linked service, but don't select a table as we'll be inputting a query instead:



Set Properties

Name: AdventureworksTables

Linked service *: LS_SQL_AdventureWorks

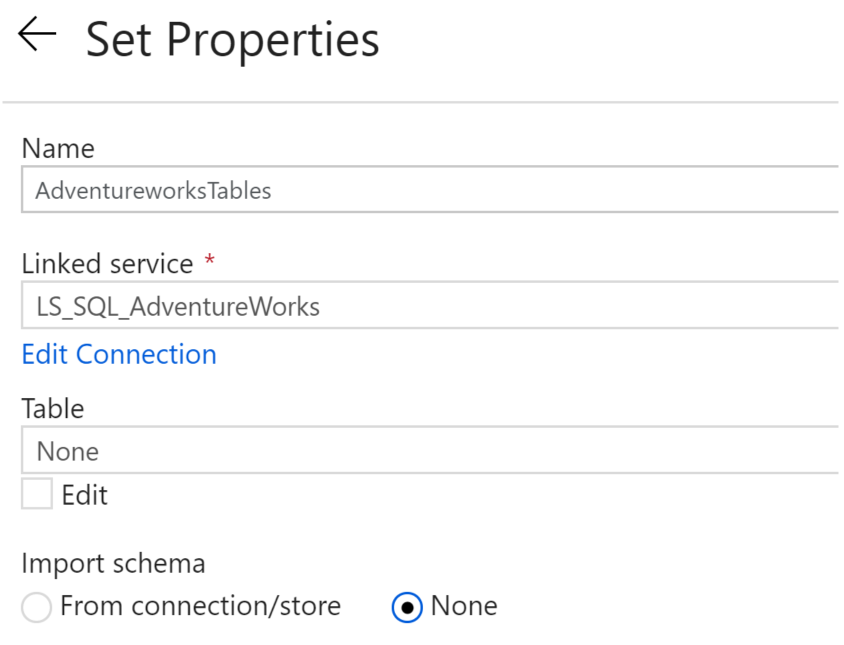
Edit Connection

Table: None

☐ Edit

Import schema: ○ From connection/store [✓ None]

Lab 03.1 – Implementing ForEach and Dynamic



← Set Properties

Name
AdventureworksTables

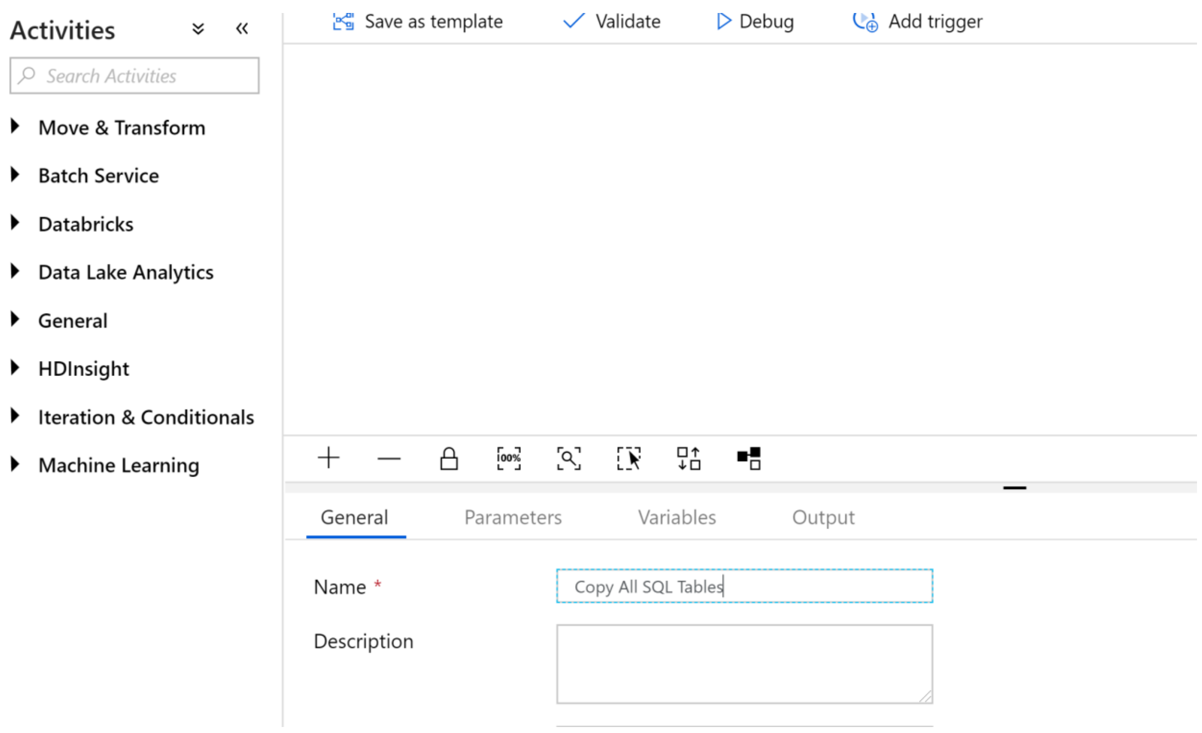
Linked service *
LS_SQL_AdventureWorks
Edit Connection

Table
None
☐ Edit

Import schema
◯ From connection/store [✓ ● None]

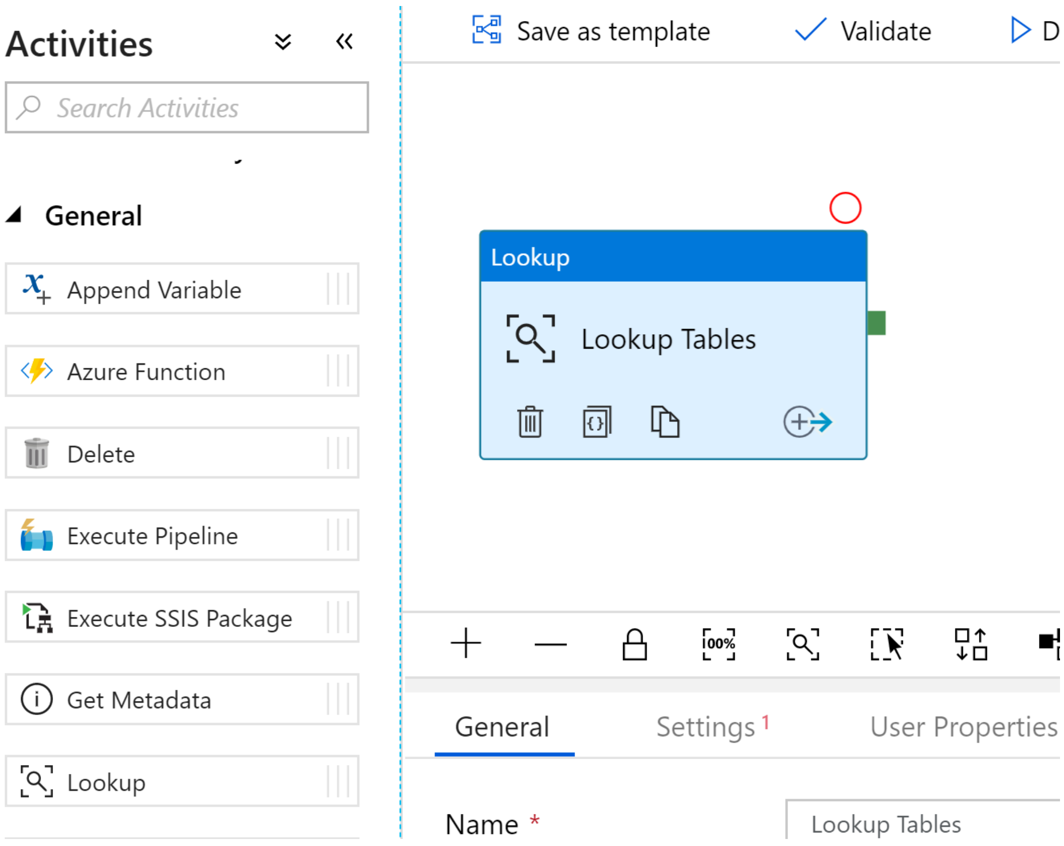
Lab 03.1.B – Create Dynamic Parent Pipeline

1. Create a new pipeline that we can use as a parent pipeline



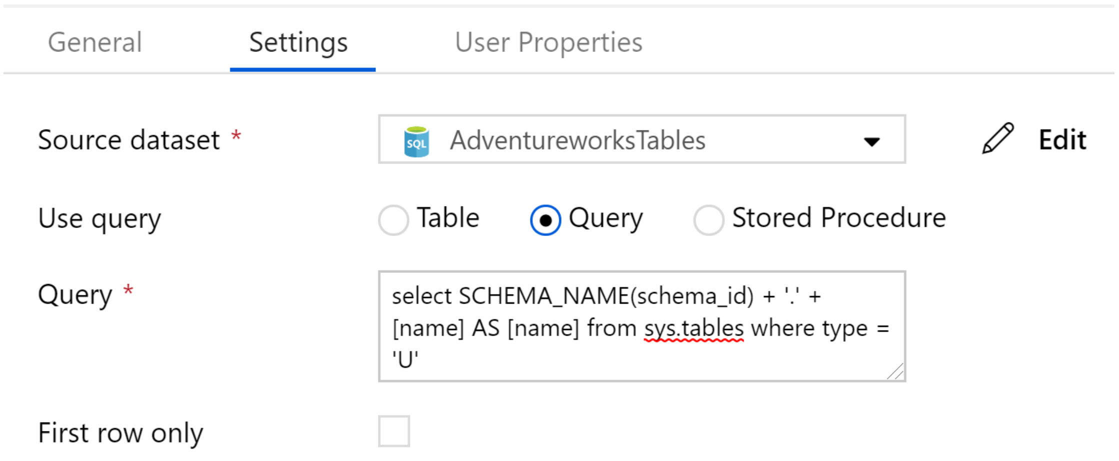
Name * Copy All SQL Tables
Description

2. Add a "Lookup" activity from the General selection



Name * Lookup Tables

3. Under “Settings”, we can pick our new AdventureworksTables dataset and input a query to return a list of table names:



Source dataset * AdventureworksTables

Use query: Table [✓ Query] Stored Procedure

Query *
```
select SCHEMA_NAME(schema_id) + '.' +
[name] AS [name] from sys.tables where type =
'U'
```

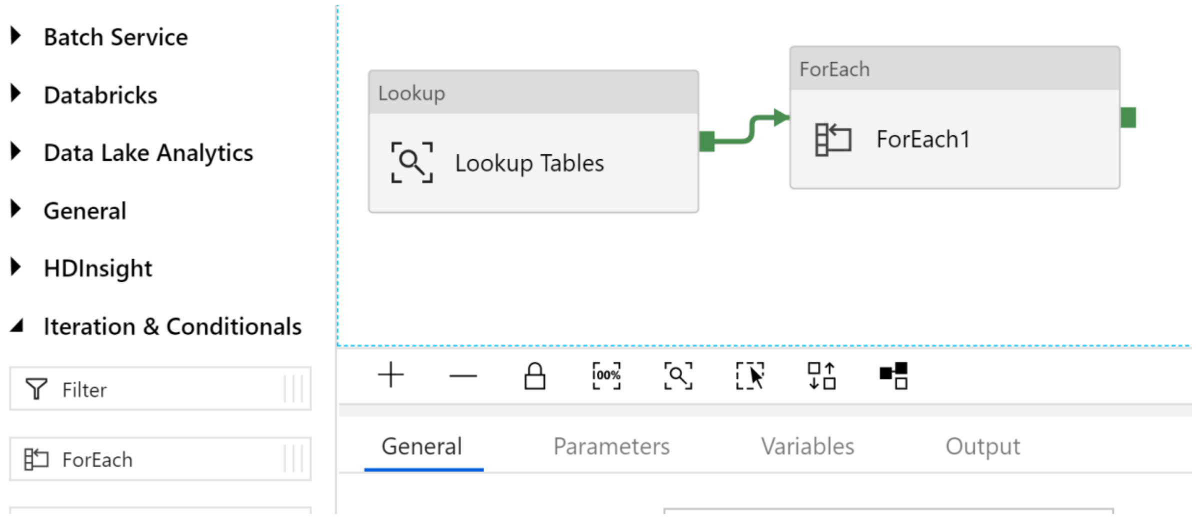
First row only ☐

If you want to copy & paste that query, it is:
select SCHEMA_NAME(schema_id) + '.' + [name] AS [name] from sys.tables where type = 'U'

Next we need to create a ForEach loop. This will execute an inner set of activities for each record in an array.

Lab 03.1 – Implementing ForEach and Dynamic

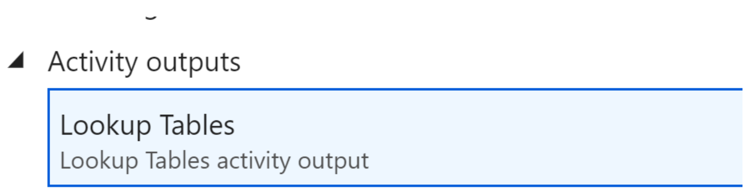
4. Add a ForEach activity from the “Iterations & Conditionals” and drag the green constraint to ensure it runs once the lookup activity has completed:



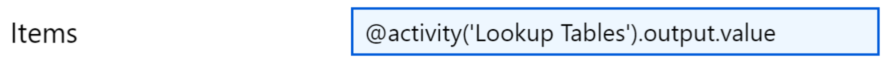
Under the “Settings” tab on the ForEach activity, this is where we tell the activity which array to loop through.

5. Click on the “Items” then on the “Add dynamic content” button

At the bottom of the dynamic content blade, you’ll see the output from the Lookup activity:



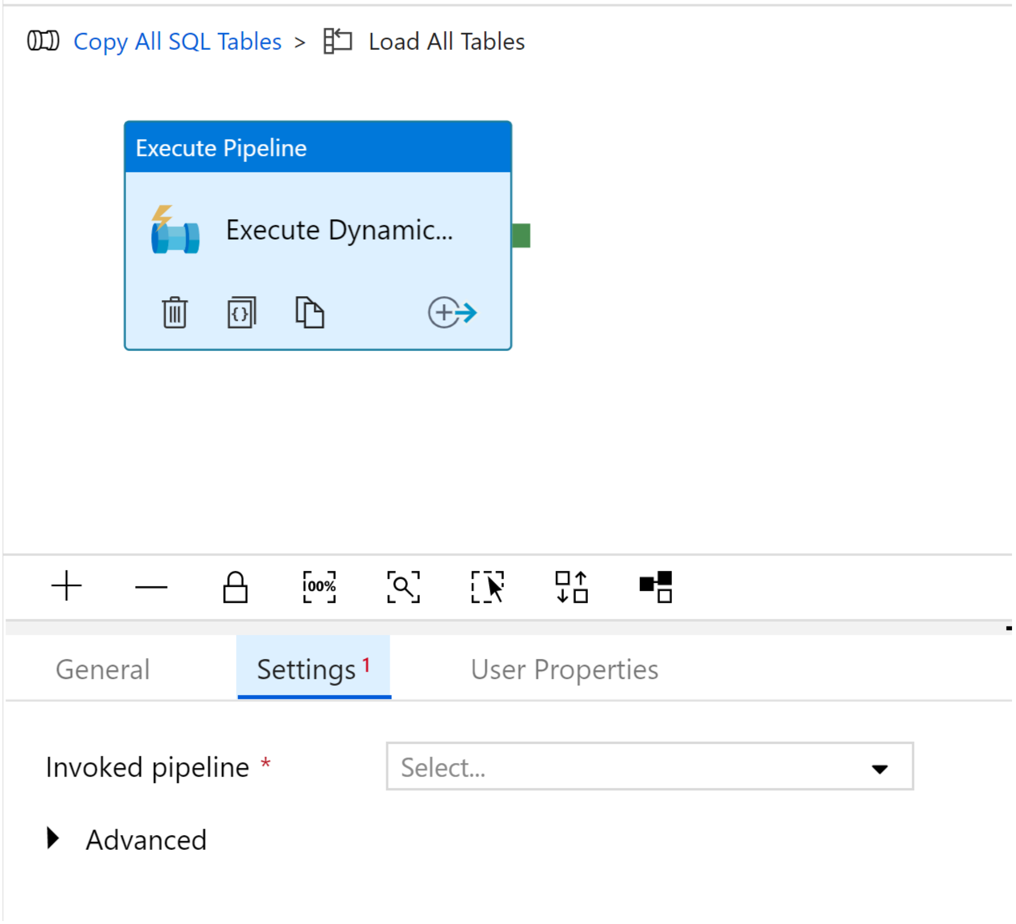
Click on this and it’ll automatically add it to your expression. The object has some additional parameters, we want the actual data, so we add “.value” to the end of the object.



Items @activity('Lookup Tables').output.value

6. Click on “Activities” then “Add Activity”. This will open another workflow plane where we can configure what happens for each activity.

Add an “Execute Pipeline” activity to this workflow


Copy All SQL Tables > Load All Tables

Execute Pipeline

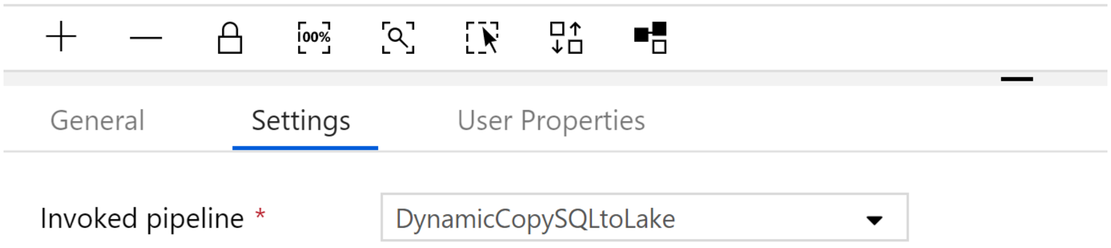
Execute Dynamic...

General | Settings 1 | User Properties

Invoked pipeline * Select...

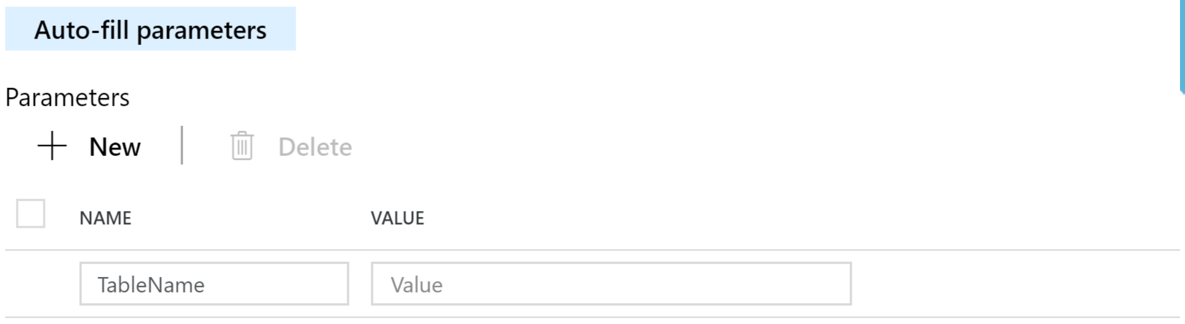
Advanced

7. Select the dynamic pipeline we made in Lab 3


General | Settings | User Properties

Invoked pipeline * DynamicCopySQLtoLake

8. Open up the Advanced tab. Click the “Auto-Fill parameters” and it should pick up our TableName parameter



Auto-fill parameters

Parameters

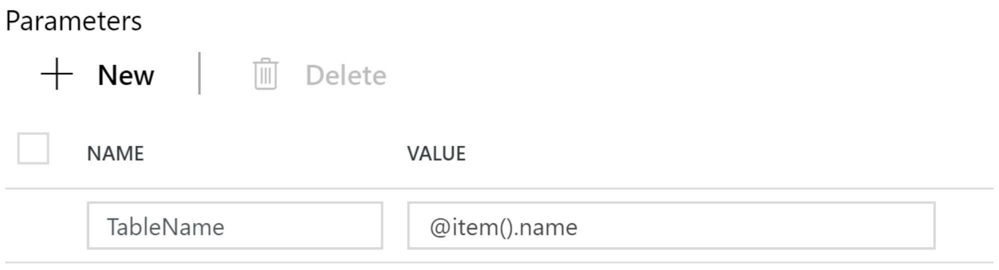
＋ New | Delete

NAME	VALUE
TableName	Value

9. Click on the “Value” and open up the dynamic content blade once more.

   There’s no GUI button for us this time – to access properties associated with the current record, we need to use the ‘@item’ object. We named the column [name] in our query, so this is now a property of the @item object.

   We can therefore use ‘@item.name’ to refer to the table name of the record we’re currently iterating through



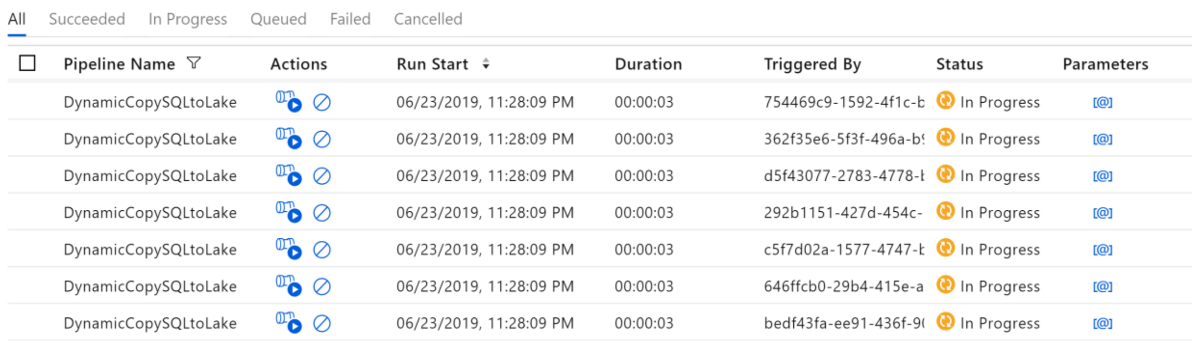
Parameters

＋ New | Delete

NAME	VALUE
TableName	@item().name

10. Now all we have to do is hit “Publish” and trigger our pipeline. Note that this time there’s no parameters to be entered as our parent pipeline is going off and getting the list of tables to execute itself.

    As we’ve used the “Execute Pipeline” activity, each of the iterations is a separate ADF pipeline that can be monitored, debugged and managed separately.

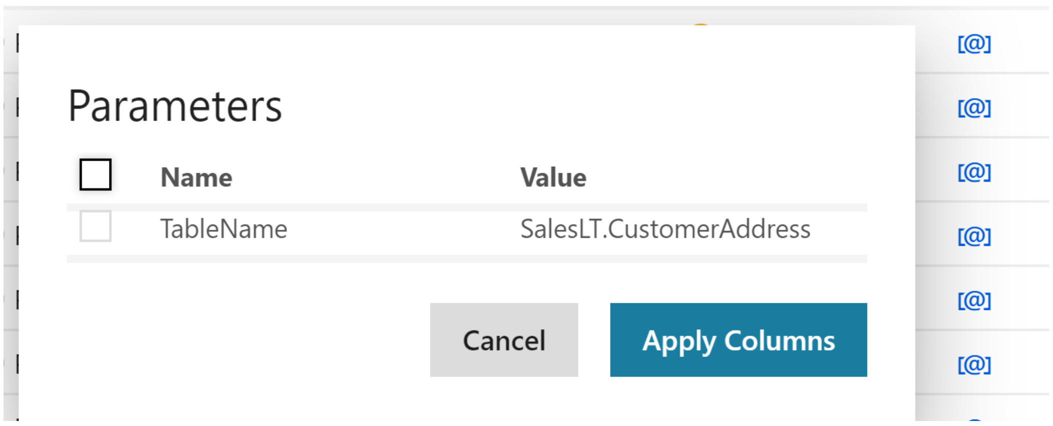


All Succeeded In Progress Queued Failed Cancelled

Pipeline Name	Actions	Run Start	Duration	Triggered By	Status	Parameters
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	754469c9-1592-4f1c-b	In Progress	[@]
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	362f35e6-5f3f-496a-b	In Progress	[@]
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	d5f43077-2783-4778-b	In Progress	[@]
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	292b1151-427d-454c-	In Progress	[@]
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	c5f7d02a-1577-4747-b	In Progress	[@]
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	646ffcb0-29b4-415e-a	In Progress	[@]
DynamicCopySQLtoLake		06/23/2019, 11:28:09 PM	00:00:03	bedf43fa-ee91-436f-9	In Progress	[@]

Lab 03.1 – Implementing ForEach and Dynamic

You can click on the “Parameters” icon for each pipeline to see which parameter has been passed into each pipeline;



Parameters

Name	Value
TableName	SalesLT.CustomerAddress