Lift and Shift of SSIS Package to Azure SSIS Runtime

Technologies showcased: Azure SSIS Runtime

Author: Michael L French

Date: 12/15/2017

Table of Contents

[Summary 3](#_Toc508285531)

[Prerequisites 4](#_Toc508285532)

[Scenario 4](#_Toc508285533)

[Part 1 – Configure Azure Data Factory SSIS Integration Runtime 4](#_Toc508285534)

[Part 2 – Upload an SSIS Package to the Integration Services Catalog 8](#_Toc508285535)

[Part 3 – Use SSMS to manually run the SSIS Package 17](#_Toc508285536)

[Part 4 – Create an Azure Data Factory Pipeline, Trigger Based Scheduled Run to execute a Stored Procedure 18](#_Toc508285537)

## Summary

In this Lab Module, we will walk through the process of configuring Azure Data Factory Integration Runtime to schedule an execution of an SSIS Package. The package will simulate a typical Data Warehouse Extract, Transform, and Load cycle.



This involves:

* Creating Azure SSIS Integration Runtime using Graphical User Interface
* Uploading an SSIS Package to the Integration Services Catalog through SSMS
* Using SSMS to manually run the package and monitor the run using a standard catalog report
* Creating an Azure Data Factory Pipeline to call a Stored Procedure to do a trigger based scheduled run of the package

## Prerequisites

* Azure Subscription with rights to use/deploy Azure services, and X of Azure Credit
* Azure Data Factory
* ADFLab.ispac (located in Lab modules/Lab 2)
* SQL Server Management Studio
* Azure Databases from Module 1:
  + AirlinePerformance-OLTP
  + AirlinePerformance-ODS
  + AirlinePerformance-Staging
  + AirlinePerformance-DW
  + Azure Blob Storage Container
* Azure SSIS Feature Pack for SQL Server 2017   
  https://www.microsoft.com/en-us/download/details.aspx?id=54798

## Scenario

|  |  |  |
| --- | --- | --- |
| Part 1 – Configure Azure Data Factory SSIS Integration Runtime | | |
| **Scenario** | | |
| In this section, we will walk through the steps to use the ADF GUI to configure SSIS Integration Runtime. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| Connect to the Azure Data Factory as set up in Module 1 called adflab-adf. | 1. Go to the ADF GUI and click Configure SSIS Integration Runtime |  |
| After clicking Configure SSIS Integration Runtime, the following dialog starts. | 1. Enter ADFLab-SSIS into the Name field and click Next |  |
| Then we enter the database connection information.  The default admin\password in the PowerShell script was labdadmin\L@bP@ss01. If you changed it put in your username\password for the SQL Server admin account. | 1. Select the Database Server Endpoint that begins with the resource group name concatenated ‘sql’ and the 5-character hash of the resource ID at the end. 2. Enter labadmin as the Admin Username. 3. Enter L@bP@ss01 for the Admin Password 4. Click Next. |  |
| Adjusting Advanced Settings is not required for this lab. | 1. Configure Advanced Settings. For this lab choose default settings and Click Finish. |  |
| The SSISDB is created by creating the service, It can take up to 30 minutes from the status to change from Starting to Running. | 1. Confirm that the Status has changed to from Starting to Running. |  |

|  |  |  |
| --- | --- | --- |
| Part 2 – Upload an SSIS Package to the Integration Services Catalog | | |
| **Scenario** | | |
| Next, we will connect to the Integration Services Catalog to upload the provided package. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| From SQL Server Management Studio, connect to the Integration Services Catalog. | 1. Connect to Azure SQL Endpoint with Management Studio. Server name is the Database Server Endpoint that begins with the resource group name concatenated ‘sql’ and the 5-character hash of the resource ID at the end plus .database.windows.net 2. Select SQL Server Authentication. 3. Enter labadmin as the Admin Username. 4. Enter L@bP@ss01 for the Admin Password 5. Click Options and enter SSISDB inn Connect to database dropdown. |  |
| Build the structure necessary to deploy a package to the Catalog. | 1. Right Click on the SSISDB in the Integration Services Catalogs and Create folder called ADFLab. |  |
|  | 1. Navigate to the Projects folder beneath the ADFLab folder. Rick click on the Projects folder to Start the Deploy Projects Wizard. |  |
|  | 1. Click Next, if the Introduction Screen is displayed. |  |
| Select the source for the SSIS Project Deployment. | 1. Select Project Deployment and click Browse locate the ADFLab.ipsac file in the adflab\Lab Modules\Lab 2 folder. |  |
| Select the Destination Integration Services Catalog | 1. Enter the Server Name that holds the SSISDB as created in Part 1. Remember to use SQL Server Authentication. 2. Click the Connect Button and verify the Path. |  |
|  | 1. Review Validate Results. |  |
| Review configurations and Deploy the SSIS Project | 1. Deploy Project. |  |
| Verify that the Project was deployed successfully. | 1. Verify Deployment Results. |  |
| We will now configure the connection strings for the services that we created in Module 1. The pattern for the name will be as follows <prefix><service><character suffix>. For example: prefixsqlnjezw or prefixstoragenjezw | 1. From the Azure Portal, retrieve the names of the SQL Server and Blob Storage Account |  |
| It is important to configure each connection string for the Entry-point packages and project. | 1. In the Integration Catalog, right click on the ADFLab Project that we just deployed and select Configure. 2. Select the Connection Managers tab. 3. For each SQL Server connection configure the connection string, server name, and password. 4. For the Azure blog storage connection you will need to put in your storage account name for the AccountName and in the ConnectionString. For the AccountKey navigate to the Storage Account blade in the Azure portal and click Access Keys and copy either key1 or key2. |  |

|  |  |  |
| --- | --- | --- |
| Part 3 – Use SSMS to manually run the SSIS Package | | |
| **Scenario** | | |
| Execute the Package manually from SQL Server Management Studio. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We will now verify that the package has been deployed properly and executes successfully. | 1. Right Click on the Package and choose Execute |  |
|  | 1. Execute Package |  |
| We can monitor the execution results from SQL Server Management Studio. | 1. Monitor Package Execution from Integration Services Dashboard |  |

|  |  |  |
| --- | --- | --- |
| Part 4 – Create an Azure Data Factory Pipeline, Trigger Based Scheduled Run to execute a Stored Procedure | | |
| **Scenario** | | |
| In this section we will create an Azure Data Factory Pipeline to schedule the execution of the uplifted SSIS Package | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| Using the ADF GUI, create a Pipeline to execute a package from the Integration Services Catalog. | 1. From the Data Integration App, click the “+” to create a new Pipeline |  |
|  | 1. Name the Pipeline Module2 and Save |  |
| Begin to build a simple pipeline with one activity. | 1. Drag a Stored Procedure Activity to the Design Surface and name it Execute SQL |  |
| The next set of steps we will walk through the detailed steps to configure the Stored Procedure Activity. | 1. Add New Linked Service named SSISDB Azure Endpoint. Select Azure SQL Database as the Type. |  |
| A single parameter containing the SQL required to create an execution instance from the Integration Services Catalog. | 1. Configure the stored procedure sp\_executesql by adding a new parameter named stmt of type string with the following value:  DECLARE @return\_value INT, @exe\_id BIGINT, @err\_msg NVARCHAR(150) EXEC @return\_value=[SSISDB].[catalog].[create\_execution] @folder\_name=N'ADFLab', @project\_name=N'ADFLab', @package\_name=N'Module2.dtsx', @use32bitruntime=0, @runinscaleout=1, @useanyworker=1, @execution\_id=@exe\_id OUTPUT EXEC [SSISDB].[catalog].[set\_execution\_parameter\_value] @exe\_id, @object\_type=50, @parameter\_name=N'SYNCHRONIZED', @parameter\_value=1 EXEC [SSISDB].[catalog].[start\_execution] @execution\_id=@exe\_id, @retry\_count=0 IF(SELECT [status] FROM [SSISDB].[catalog].[executions] WHERE execution\_id=@exe\_id)<>7 BEGIN SET @err\_msg=N'Your package execution did not succeed for execution ID: ' + CAST(@exe\_id AS NVARCHAR(20)) RAISERROR(@err\_msg,15,1) END |  |
| Verify that the Stored Procedure Activity has been configured properly. | 1. Validate the Pipeline |  |
| Once the Pipeline has passed validation, test the actual execution to ensure that the Module2.dstx package has executed.  We can monitor the execution results from MS SQL Server Management Studio as in Part 3 using the Integration Services Dashboard. | 1. Execute a Test Run |  |
| Now that we have successfully configured and tested the Pipeline, we can begin to create an automated execution schedule.                                       When you click Next the following reminder appears. | 1. Add a New Trigger named Module2 Trigger. Use the Recurrence of Hourly.           Note: Please set the start date to a few minutes ahead of the current time inn order to observe the execution. |  |
| From the Data Integration App, use a two-step process to Sync and Publish our changes. | 1. Sync your changes |  |
| Ensure that no conflicts exist in our code. | 1. Verify the changes and click Sync                                Note: Uncheck Publish changes after sync. |  |
| Once we click publish, the Trigger is live, and the Pipeline will begin to execute on the frequency configured. | 1. Publish your Changes |  |
|  | 1. Verify that Trigger is Scheduled |  |
| Validate that the Pipeline has executed using the Portal. | 1. Monitor Pipeline Runs from Data Integration App |  |
| After the completion of all steps in this module, remember to deactivate the Trigger to stop further executions | 1. Deactivate the Trigger. 2. Sync your changes. 3. Publish your changes. |  |

**IMPORTANT: AVOID INCURRING EXTRA CHARGES BY PAUSING YOUR SUBSCRIPTION RESOURCES**