Lab: Extract and Load with ADF

Technologies showcased: ADF GUI, S3, SQL Azure and Azure Storage, Pipeline Parameters, Triggers

Table of Contents

[Summary 3](#_Toc504754934)

[Pre-requisites 4](#_Toc504754935)

[Scenario 4](#_Toc504754936)

[Part 1 – Create an Azure Data Factory (v2) 4](#_Toc504754937)

[Part 2 – Connect ADF to a code repository to begin using the ADF GUI (Optional) 6](#_Toc504754938)

[Part 3 – Setting up the Connections in the ADF GUI (S3 -> Blob) 10](#_Toc504754939)

[Part 4 – Setting up the Copy Activity in the ADF GUI (S3 -> Blob) 17](#_Toc504754940)

[Part 5 – Setting up the Connections in the ADF GUI (Azure SQL Database -> Blob) 22](#_Toc504754941)

[Part 6 – Setting up the Copy Activity in the ADF GUI (Azure SQL Database -> Blob) 26](#_Toc504754942)

[Part 7 – Using Parameters and Triggers (scheduling) in ADF GUI 31](#_Toc504754943)

## Summary

This tutorial walks through creating a pipeline copy activity to copy a file from a S3 storage location to an Azure blob storage container, so we can prepare the file to be processed later for transformation.

In this lab we will:

* Show the graphical user interface for creating a pipeline
* Copy CSV file via a Copy Activity
* Copy Structed data from SQL Azure via a Copy Activity
* Use parameters to make the pipeline easy to change and more reusable

## Pre-requisites

* Azure Subscription with rights to use/deploy Azure services, and X of Azure credit
* Azure Blob storage container
* Azure Data Factory
* Visual Studio Team Services Git project (optional)

## Scenario

|  |  |  |
| --- | --- | --- |
| Part 1 – Create an Azure Data Factory (v2) | | |
| **Scenario** | | |
| We are going to use the portal to create the Azure Data Factory. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
|  | 1. Go to Azure Portal- <https://portal.azure.com>. 2. Select **New** on the left menu, select **Data + Analytics**, and then select **Data Factory**. | Data Factory selection in the "New" pane |
|  | 1. On the **New data factory** page, enter **adflab-adf** for **Name.** 2. Make sure you select Version as ‘V2 (Preview)’   **Note**: The name of the Azure data factory must be globally unique. Please modify the name if the Name validation fails. |  |
|  | 1. After the creation is complete, you see the **Data Factory** page. Select the **Author & Monitor** tile to start the Azure Data Factory user interface (UI) application on a separate tab. | Home page for the data factory, with the "Author & Monitor" tile |

|  |  |  |
| --- | --- | --- |
| Part 2 – Connect ADF to a code repository to begin using the ADF GUI (Optional) | | |
| **Scenario** | | |
| One option to be able to sync our code is to connect ADF to a code repository. This section walks through the steps to connect ADF to a Visual Studio Team Services Git project, so we can save our code for later re-use. Note that this is not required but a recommended best practice. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| Please use the Azure Data Factory we created in **Part 1**. | 1. Navigate to the Azure portal within your web browser and navigate to <https://portal.azure.com> 2. Open the Azure Data Factory blade **adflab-adf**. | *TO-DO: Replace with GUI Pipeline editor tile exists* |
|  | 1. In the Overview Blade, Click on ‘**Author and Monitor**’ |  |
| We can connect to a code repository before we start creating pipelines using the GUI in order to version our code. (This is optional, and you can work with GUI without configuring Code repository)  ***NOTE****: This is only visible when you have not Git configured.* | 1. Click the Configure Code repository button to begin connecting this Azure data factory to a code repository. |  |
| Use an existing VSTS account or create a new one to configure Code Repository. | 1. You can either create a new VSTS account for this lab or use an existing one. You can create a new one - [here](https://app.vsaex.visualstudio.com/profile/account) and configure it in ADF GUI under repository settings. You need to create/ select a Project under this Account.      1. The Repository Settings pane will appear on the right. 2. Click the Finished button when you have verified your settings. |  |

|  |  |  |
| --- | --- | --- |
| Part 3 – Setting up the Connections in the ADF GUI (S3 -> Blob) | | |
| **Scenario** | | |
| We now want to use the GUI to create a pipeline to copy the CSV files from S3 to Azure blob storage to be ready for transformation. Our first step is setting up the connections and linked services need for the source and destination. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We will be using the previously configured code repository in ADF (if you finished the Part 2). Otherwise you can still create Pipeline without a code repository. | 1. Launch the ADF GUI from Azure Portal.      1. Click the Create Pipeline button under Let’s Get Started. |  |
|  | 1. You should now see Pipeline GUI editor. |  |
|  | 1. Fill out the pipeline name. We will be using **CopyPipeline** as our pipeline name. 2. Click the Save Icon on top of the screen. |  |
| We now need to setup our Connections i.e. Linked Services to for both the source (S3) and destination (Azure blob storage) | 1. In the Left Menu click the Connections menu item. 2. Click the +New button under Linked Services. |  |
| Amazon S3. | 1. In the right pane you should now see the list of possible Linked Services. 2. Click on the **Amazon S3** tile. 3. Click **Continue**. |  |
|  | 1. In the right pane you should see the properties to configure the Amazon S3 account link.   We will name this linked service **AmazonS3-Source** and using the Default runtime. Use the following for the  **Access Key ID:**  AKIAI7PF6C2LJQ2X75VA and  **Secret Access Key:**  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   1. Click the Test Connection to verify settings are entered correctly. 2. Click Save. |  |
| Now we need to add destination, Azure storage. | 1. Click the +New button under Linked services. 2. In the right pane you should now see the list of possible Linked Services. 3. Click on the **Azure Storage** tile. 4. Click **Continue**. |  |
|  | 1. In the right pane you should see the properties to configure the Azure Storage account link. 2. We will name this linked service **AzStorage-Staging** and using the Default runtime. For the Account selection method use **From Azure subscription** and then choose your subscription used for these labs and the storage account created for the Staging area ***adflabstaging.***   *Note: You may create a new storage account or use an existing one.*  *Additionally, create a blob container named ‘****inputs3’*** *and* ***'inputsql'*** *in this storage for later usage. In this case we have used* ***adflabstaging*** *as storage account name.*   1. Click the **Test Connection** to verify settings are entered correctly. 2. Click **Save**. |  |

|  |  |  |
| --- | --- | --- |
| Part 4 – Setting up the Copy Activity in the ADF GUI (S3 -> Blob) | | |
| **Scenario** | | |
| We now want to use the GUI to create a Copy Activity in the pipeline to move the files from the S3 source to our Azure storage destination. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We will be using the previously created pipeline called CopyPipeline. | 1. Click the **CopyPipeline** in the left menu to return to the pipeline GUI. |  |
|  | 1. In the Pipeline GUI, drag the **Copy activity** (under DataFlow) to the empty pane above General. 2. Rename the activity to **S3toAzureBlob**. 3. Click **Save**. |  |
|  | 1. Click the Source Tab in the Copy Activity GUI. 2. Click the +New button next to Source Dataset. |  |
|  | 1. You should now see the list of source dataset connectors. 2. Choose the **Amazon S3** **dataset** and click **Finish.** |  |
| We will be using the Linked Service we created earlier. | 1. You should now add the connection property information. 2. Name this **datasetS3** 3. Fill out the following information: **Linked Service** -> **AmazonS3-Source**, **File Path** -> Click the Browse button and drill down to the **adflab folder** (double click on folder) and the **master.txt** file, check the **‘Binary Copy’** checkbox.   ***Note****: Typically, you would select ‘****Binary copy’*** *while copying files as is.* |  |
|  | 1. Click back on the **CopyPipeline**. 2. Click the **S3toAzureBlob** copy activity. 3. Click the Sink Tab in the Copy Activity GUI. 4. Click the +New button next to Source Dataset. |  |
|  | 1. You should now see the list of sink dataset connectors. 2. Choose the **Azure Blob storage dataset** and click **Finish**. |  |
| We will be using the Azure blob storage linked service we created in the previous section. | 1. Name the dataset as **datasetBlob** 2. Fill out the following information**: Linked Service** -> **AzStorage-Staging**, **File Path** -> **inputs3** Check ‘**Binary Copy’** checkbox.   **Note**: Typically, you would select ‘**Binary copy’** while copying files as is. |  |
| Now we can test our copy activity from within the ADF GUI editor. | 1. Navigate to the **CopyPipeline**. Click the **Test Run** icon at the top menu to test and run the copy activity. |  |

|  |  |  |
| --- | --- | --- |
| Part 5 – Setting up the Connections in the ADF GUI (Azure SQL Database -> Blob) | | |
| **Scenario** | | |
| We now want to use the GUI to create another copy activity in the same pipeline to copy the Data from Azure SQL DB to Azure blob storage to be ready for transformation along with the earlier CSV file. Our first step is setting up the connections and linked services need for the source and destination. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We now need to setup our Connections i.e. Linked Services to for both the source (SQL Azure) and destination (Azure blob storage) | 1. In the Left Menu click the Connections menu item. 2. Click the +New button under Linked Services. |  |
| Azure SQL DB | 1. In the right pane you should now see the list of possible Linked Services. 2. Click on the **Azure SQL Database** tile. 3. Click **Continue**. |  |
| We will use a predefined Azure SQL Database for this lab. | 1. In the right pane you should see the properties to configure the Azure SQL Database account.   We will name this linked service **AzureSqlDatabase-Source** and using the Default runtime. Use the following  *Account Selection Method ->* ***Manual****,*  *Fully qualified domain name ->*  ***adlabserver.database.windows.net,***  *Database name ->* ***adflab****,*  *User name ->* ***lab\_user****,*  *Password ->* ***P@ssw0rd***   1. Click the **Test Connection** to verify settings are entered correctly. 2. Click **Save**. |  |
| Since the sink/ destination storage is the same as for the previous copy activity, we do not need to create the connection again. |  |  |

|  |  |  |
| --- | --- | --- |
| Part 6 – Setting up the Copy Activity in the ADF GUI (Azure SQL Database -> Blob) | | |
| **Scenario** | | |
| We now want to use the GUI to create a Copy Activity in the pipeline to move the files from the Azure SQL Database as source to our Azure storage destination. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We will be using the previously created pipeline called CopyPipeline.  We will add a new copy activity. | 1. Click the **CopyPipeline** in the left menu to return to the pipeline GUI. |  |
|  | 1. In the Pipeline GUI, drag the **Copy activity** (under DataFlow) to the empty pane above General. 2. Rename the activity to **AzureSQLtoAzureBlob**. 3. Click **Save**. |  |
|  | 1. Click the Source Tab in the Copy Activity GUI. 2. Click the +New button next to Source Dataset. |  |
|  | 1. You should now see the list of source dataset connectors. 2. Choose the **Azure SQL dataset** and click **Finish.** |  |
| We will be using the Linked Service we created earlier. | 1. You should now add the connection property information. 2. Name this **datasetAzureSqlTable** 3. Select Table -> **[SalesLT].[Customer]** 4. Click **Preview Data** to preview the first several data rows. |  |
| We will only copy the required columns from source to sink, and refrain from copying sensitive columns by using a query. | 1. Click on **CopyPipeline** and then click **AzureSQLtoAzureBlob** Activity. 2. Select ‘**Source’** -> Use Query -> **Query**  **SELECT CustomerID, CompanyName, SalesPerson, ModifiedDate**   **FROM [SalesLT].[Customer]**  ***Note****: This Query may change based on your table selection.*   1. Click on **Preview data** to ensure the query works. |  |
|  | 1. Click back on the **CopyPipeline**. 2. Click the **AzureSQLtoAzureBlob** copy activity. 3. Click the Sink Tab in the Copy Activity GUI. 4. Click the +New button next to Source Dataset. |  |
|  | 1. You should now see the list of sink dataset connectors. 2. Choose the Azure Blob storage dataset and click **Finish**. |  |
| We will be using the Azure blob storage linked service we created in the previous section for this dataset. | 1. Name the dataset as **datasetBlobfromSQL** 2. Fill out the following information: **Linked Service** -> **AzStorage-Staging**, **File Path** -> Click the Browse button and drill down to the **inputsql** container. **Note**: Make sure **inputsql** container exists or else create one first. 3. **Save** the dataset |  |
| Now we can test our copy activity from within the ADF GUI editor. | 1. Navigate to the **CopyPipeline** 2. Click the **Test Run** icon at the top menu to test and run our copy activity. 3. Final **CopyPipeline** looks like this |  |
| You can use Storage Explorer or use Azure Portal to verify the data ingestion into appropriate storage containers. |  |  |

|  |  |  |
| --- | --- | --- |
| Part 7 – Using Parameters and Triggers (scheduling) in ADF GUI | | |
| **Scenario** | | |
| You can define parameters at the pipeline level and pass arguments while you're invoking the pipeline on-demand or from a trigger. Activities can consume the arguments that are passed to the pipeline. Using parameters, you can build more flexible pipelines.  And triggers can be used to execute the pipelines on a schedule or on-demand. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We will use a parameter and use it to name the output dataset in sink. | 1. Navigate to **CopyPipeline** -> **Parameters**. Add new parameter. 2. Name it as **filename,** let the **Value** be empty. 3. Click **Save** |  |
|  | 1. Navigate to **datasetBlobfromSQL** -> **Parameters** -> **File Name**, and set the value as **@pipeline().parameters.filename** |  |
|  | 1. Click on **Connection** tab to verify the parameters |  |
|  | 1. Navigate to **datasetBlob** -> **Parameters** -> **File Name**, and set the value as **@pipeline().parameters.filename** |  |
|  | 1. Click on **Connection** tab to verify the parameters |  |
|  | 1. Navigate to the **CopyPipeline,** do a **Test Run.** 2. It will ask for an **input parameter**. Enter appropriate name and this will be used as the file name in sink. |  |
| Publish Code Repository (OPTIONAL) only if you had configured it in Step 2. | 1. Click on **Sync** (and **Publish**). This will write the changes to Master. |  |
| We can configure trigger for operationalizing pipelines. | 1. Click on **CopyPipeline** -> **Triggers** -> Add new trigger. 2. Enter the trigger properties accordingly. In this case, we create a daily **tumbling window trigger**.   Set **Start**, **End time** for Trigger. Check **Activated** check-box.   1. Click **Next**. |  |
| Setting system variables to create partitions in the sink during operationalized copy pipeline runs. | 1. In the Trigger Run Parameter window,   Set **fileName** -> **copyfromsql\_@{formatDateTime(trigger().outputs.windowStartTime, 'yyyy-MM-dd')}**  **Note**: Expressions can be changed based on requirements.   1. Click **Finish**.   **Make sure you ‘Publish’ for the trigger to activated.** |  |
|  | 1. Navigate to **Monitoring** section to see **pipeline runs**. 2. We can find the appropriate **parameters** being passed during each triggered run. |  |
|  | 1. On Successful run of the **CopyPipeline**, navigate to the storage locations using **Storage Explorer** or **Azure Portal** (Storage Account), to verify the files copied. The filename would be defined by the **parameter** -> **fileName**. |  |

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