Azure Data Factory Hands-On Mini-Project

Developing and end-to-end pipeline

Alfred Hull

2021

# **Data Engineering Lab.**

## **Lab 1. Prepare the Environment**

## **Lab 2. Linked Services Datasets and Integration Runtimes**

## **Lab 3. Copy Activity Parameters Debug and Publishing**

## **Lab 4. Lookup activity For each loop and Execute Pipeline activity**

## **Lab 5. Get Metadata activity filter activity and complex expressions**

## **Lab 6. Self-hosted Integration Runtime – decompress files and Delete activity**

# **Azure Data Factory Mapping Data Flows.**

## **Lab 7. SmartFoodsCustomerELT**

## **Lab 8. ELT with Mapping Data Flows – Practice exercises**

# **Deliverable Questions:**

## **Question 1. Why should one use Azure Key Vault when working in the Azure environment? What are the pros and cons? What are the alternatives?**

### Q1. Response:

Centralizing storage of application secrets in Azure Key Vault allows you to control their distribution. Key Vault greatly reduces the chances that secrets may be accidentally leaked. When using Key Vault, application developers no longer need to store security information in their application. Not having to store security information in applications eliminates the need to make this information part of the code. For example, an application may need to connect to a database. Instead of storing the connection string in the app's code, you can store it securely in Key Vault. Your applications can securely access the information they need by using URIs. These URIs allow the applications to retrieve specific versions of a secret. There is no need to write custom code to protect any of the secret information stored in Key Vault.

Pros:

High Level of Security

The most valuable feature is that you can retrieve user account details from the cloud.

Ease of use and enabling clients to manage keys.

Cons:

Azure needs to provide versions of Key Vault that are suitable for different sizes of companies.

The initial setup could be less complex for first-time users

Overhead of managing Access Control List every time a new developer joins the team.

Alternatives are AWS Key Management Service (AWS), HashiCorp Vault etc.

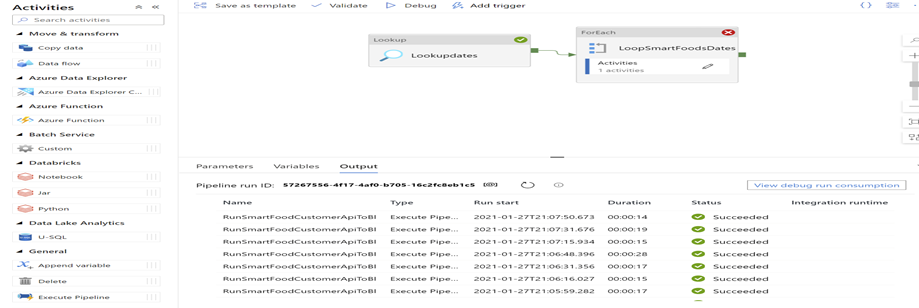
## **Question 2. How do you achieve loop functionality within a Azure Data Factory pipeline? Why would you need to use this functionality in a data pipeline?**

### Q2. Response:

The ForEach Activity defines a repeating control flow in your pipeline. This activity is used to iterate over a collection and executes specified activities in a loop. The loop implementation of this activity is like the Foreach looping structure in programming languages.

In the below screenshot ‘Foreach’ activity loops through the result of the lookup ‘Lookupdates’. On the Settings tab, you can provide the items you want to loop through. In our case the output of the preceding Lookup activity from the previous step: @activity('LookupDates’).output.value

On the Activities tab, we can add a new activity that we want to execute for all the Dates. The Activity, in this case, is ‘RunSmartFoodCustomerApitoBlob’

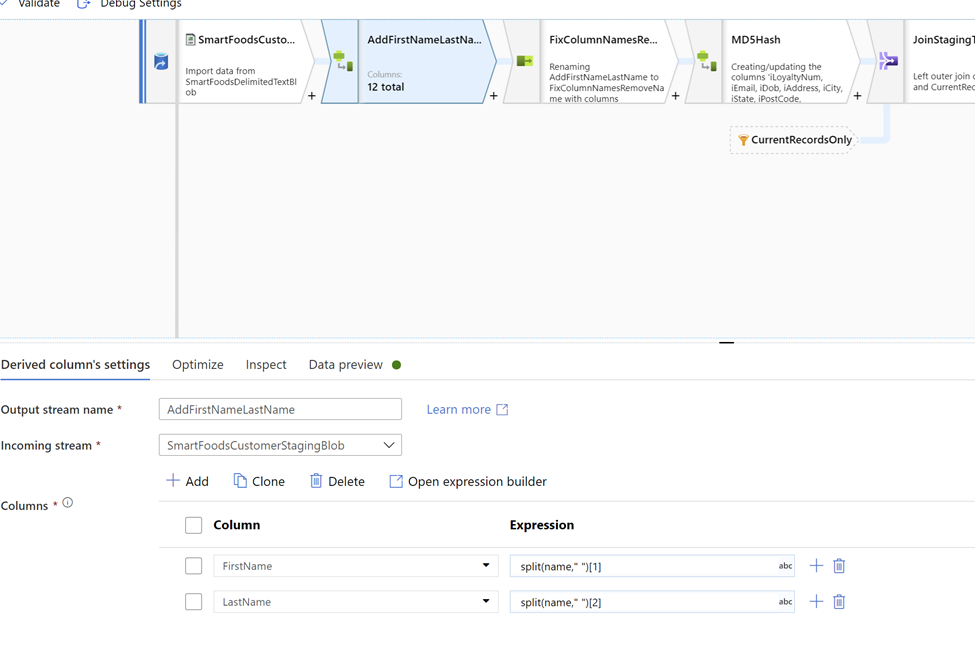


## **Question 3. What are expressions in Azure Data Factory? How are they helpful when designing a data pipeline? Please explain with an example.**

### Q3. Response:

In the mapping data flow, many transformation properties are entered as expressions. These expressions are composed of column values, parameters, functions, operators, and literals that evaluate to a Spark data type at run time. Mapping data flows has a dedicated experience aimed to aid you in building these expressions called the Expression Builder. Utilizing IntelliSense code completion for highlighting, syntax checking, and autocompleting, the expression builder is designed to make building data flows easy.

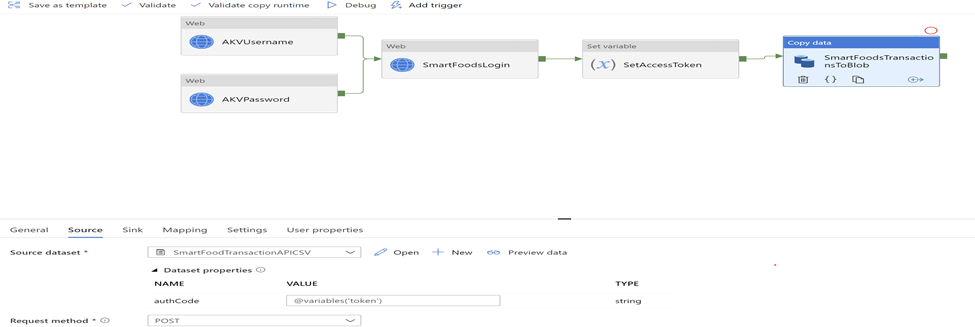
For Example Below expression splits the name column from Incoming Stream ‘SmartFoodsCustomerStagingBlob’ into firstName and lastName columns



## **Question 4. What are the pros and cons of parametrizing a dataset’s activity in Azure Data Factory?**

### Q4. Response:

Below Copy Activity parametrizes the dataset property ‘authCode’ to take the value generated by previous activity.



The advantage of this approach is that the pipeline is fully automated and avoids the use of hardcoded values anywhere in the pipeline.

Data analysis is important step to be performed before parametrizing the dataset properties.

## **Question 5. What are the different supported file formats and compression codecs in Azure Data Factory? When will you use a Parquet file over an ORC file? Why would you choose an AVRO file format over a Parquet file format?**

### Q5. Response:

Azure Data Factory supports the following file formats.

• Avro format

• Binary format

• Delimited text format

• Excel format

• JSON format

• ORC format

• Parquet format

• XML format

Parquet file has the following compression-related options: NONE, SNAPPY, GZIP, and LZO

ORC file has three compression-related options: NONE, ZLIB, SNAPPY

Other compression-related options are ZipDeflate, Deflate etc

When to use Parquet file over ORC file?

Parquet is a row columnar data format created by Cloudera and Twitter in 2013. Parquet files consist of row groups, header, and footer, and in each row group data in the same columns are stored together. Parquet is specialized in efficiently storing and processing nested data types. If you have a lot of complex nested columns in your dataset and often only query a subset of the subcolumns, Parquet would be a good choice. Parquet is implemented using the record shredding and assembly algorithm described in the Dremel paper, which allows you to access and retrievesubcolumns without pulling the rest of the nested column.

One should consider the platform/framework you are using when choosing a data format, as data formats perform differently depending on where they are used. ORC works best with Hive (since it is made for Hive). Spark provides great support for processing Parquet formats.

When to use Avro file over Parquet file?

One challenge of handling big data is the frequent changing of data schema: e.g. adding/dropping columns and changing columns names. If your data schema changes a lot and you need high compatibility for your old/new applications, Avro is a good choice over Parquet. Plus, Avro’s data schema is in JSON and Avro can keep data compact even when many different schemas exist. Also, Avro is often a good choice for Kafka.