Azure Data Factory Discussion Questions

Why should one use Azure Key Vault when working in the Azure environment? What are the alternatives to using Azure Key Vault? What are the pros and cons of using Azure Key Vault?

Azure Data Vault provides a means to handle secrets/keys/certificates used as credentials for accessing items such as databases and APIs in a secure manner. Rather than explicitly referencing these in code, these are referenced programmatically and are thus obscured from any users viewing a script. The benefit of this is a layer of security when code is pushed as this minimizes the security risk of these secrets being viewed by those who should not have access to them. Azure Data Vault is particularly useful in settings where an organization is already using Azure for its data solutions. Conversely, this may not as readily integrate in situations with mixed solutions (multiple cloud tools, mix of cloud and on-prem tools). In addition, the cost of this tool may be a challenge if budget is a concern. Some alternatives may include tools like HashiCorp Vault or environment variables/.env files for simple applications.

How do you achieve the loop functionality within an Azure Data Factory pipeline? Why would you need to use this functionality in a data pipeline?

Loop functions in Azure Data Factory accomplish similar tasks as for/while loops would in a scripting language. In ADF, these are implemented by means of ForEach (for loops) and Until (while loops) activities. For instance, you might have a situation in which a source of data consists of several partitioned flat files that must be read in sequentially, which could be handled with a ForEach activity. An Until activity could be used in other circumstances. For instance, we may receive a large flat file at a specific time each week at a variable time in a landing folder. We may want to run a task each week that continues to check this folder for ingestion until the file loads or until some timeout is reached.

What are expressions in Azure Data Factory? How are they helpful when designing a data pipeline (please explain with an example)?

Expressions allow for data pipelines in ADF to dynamically reference values while performing its task. These can even come from evaluating different parts of the upstream pipeline or by evaluation of functions. For instance, we could use expression syntax to

reference the date and timestamp when a .csv file is created from a pipeline and apply that to the filename. Another use case could be if we are tasked with managing several pipelines for a marketing program. Suppose we must ensure each pipeline for each campaign must follow global suppression rules that could be changed from time to time. Rather than having to hard code these rules into each pipeline, we could provide an expression that draws from a lookup table that includes all these rules. If these rules change, we will only need to adjust one file. All our pipelines would then evaluate this at runtime and accurately reflect the changes we have made.

What are the pros and cons of parametrizing a dataset in Azure Data Factory pipeline's activity?

Parametrizing a dataset using tools like expressions mentioned above allows for added automation, scaling, and reusability of pipeline activities. A driver of this benefit is eliminating the need to hardcode values to identify datasets, but rather use expressions to do so. For instance, suppose we had a large repository of flat files that needed to be ingested, with month/day/year information in their file names. If we needed to only ingest files from a certain month, we could parameterize the filenames and easily filter the list of files needed without having to do this manually. Some drawbacks to this might include the need for additional setup and expertise in using these types of expressions, as well as some complexity in debugging as the engineer must review these parameters to fully understand what is being referenced by them.

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?

ADF affords support of delimited text files (.txt, .csv, etc.), JSON, ORC, AVRO, XML, and XLS/XLSX formats. These can be compressed using SNAPPY, ZIP, GZIP, BZIP2, and Deflate. Parquet and ORC files are columnar oriented storage types, suited for reading columns from large datasets. In general, Parquet files will have better performance in Spark environments, while ORC will have better performance in Hive. In contrast to these, AVRO is a row-oriented format that favors streaming processes rather than large scale analytic storage seen in ORC/Parquet use cases.