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

*Azure Key Vault (AKV) is used to store all credentials for services that ADF will connect to. This has multiple advantages:*

* *Security of storing sensitive information in credentials store which only the ADF service or Administrators can read from*
* *If Credentials need to be rotated ADF Linked Service will not need to be modified*
* *When we migrate the ADF pipeline from Dev to Test to production no change is necessary*

*Disadvantage: This service is not free*

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

*Loop Functionality is achieved by using the “ForEach” Loop Activity. The input to ForEach should be the list of items to iterate over.This could be an array of items stored in variable or read from the file.*

*This could be useful in scenarios wherein same pipeline is to be executed for different dates or parameters.*

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

*Expression is JSON based formula , which allows for modification of variables or any parameter for pipeline , action or condition.*

*JSON values in the definition can be literal or expressions that are evaluated at runtime.*

*Expressions can appear anywhere in a JSON string value and always result in another JSON value. If a JSON value is an expression, the body of the expression is extracted by removing the at-sign (@). If a literal string is needed that starts with @, it must be escaped by using @@.*

*Expressions can also appear inside strings, using a feature called string interpolation where expressions are wrapped in @{ ... }. For example: "name" : "First Name: @{pipeline().parameters.firstName} Last Name: @{pipeline().parameters.lastName}"*

*Using string interpolation, the result is always a string. Say I have defined myNumber as 42 and myString as foo:*

| ***TABLE 2*** | |
| --- | --- |
| ***JSON value*** | ***Result*** |
| *"@pipeline().parameters.myString"* | *Returns foo as a string.* |
| *"@{pipeline().parameters.myString}"* | *Returns foo as a string.* |
| *"@pipeline().parameters.myNumber"* | *Returns 42 as a number.* |
| *"@{pipeline().parameters.myNumber}"* | *Returns 42 as a string.* |
| *"Answer is: @{pipeline().parameters.myNumber}"* | *Returns the string Answer is: 42.* |
| *"@concat('Answer is: ', string(pipeline().parameters.myNumber))"* | *Returns the string Answer is: 42* |
| *"Answer is: @@{pipeline().parameters.myNumber}"* | *Returns the string Answer is: @{pipeline().parameters.myNumber}* |

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

*Parameters are simply input value for operations in Data Factory . Each action has set of predefined parameters that needs to be supplied. Additionally, some blocks like pipeline and datasets allow to define custom parameters.*

* *Parameters can be used in following scenarios:*
* *Passing input file name*
* *Dynamic output filename*
* *Appending dates*
* *Changing connection parameters*
* *Conditional Programming*

The only disadvantage we can think of is parameters value can be sometimes complex to be specified in form of expression.

1. **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 form**

*Azure Data Factory supports the following file formats*

* *Avro format- The compression codec to use when writing to Avro files. When reading from Avro files, Data Factory automatically determines the compression codec based on the file metadata.  
  Supported types are "****none****" (default), "****deflate****", "****snappy****". Note currently Copy activity doesn't support Snappy when read/write Avro files.*
* *Binary format- The compression codec used to read/write binary files.  
  Allowed values are****bzip2****,****gzip****,****deflate****,****ZipDeflate****,****Tar****, or****TarGzip****.*
* *Delimited text format-The compression codec used to read/write text files.  
  Allowed values are****bzip2****,****gzip****,****deflate****,****ZipDeflate****,****TarGzip****,****Tar****,****snappy****, or****lz4****. Default is not compressed.****Note****currently Copy activity doesn't support "snappy" & "lz4", and mapping data flow doesn't support "ZipDeflate", "TarGzip" and "Tar".*
* *Excel format-The compression codec used to read/write JSON files.  
  Allowed values are bzip2, gzip, deflate, ZipDeflate, TarGzip, Tar, snappy, or lz4. Default is not compressed.*
* *JSON format- The compression codec used to read/write JSON files.  
  Allowed values are****bzip2****,****gzip****,****deflate****,****ZipDeflate****,****TarGzip****,****Tar****,****snappy****, or****lz4****. Default is not compressed.*
* *ORC format- When reading from ORC files, Data Factories automatically determine the compression codec based on the file metadata.  
  Supported types are****none****,****zlib****,****snappy****(default), and****lzo****. Note currently Copy activity doesn't support LZO when read/write ORC files.*
* *Parquet format- When reading from Parquet files, Data Factories automatically determine the compression codec based on the file metadata.  
  Supported types are "****none****", "****gzip****", "****snappy****" (default), and "****lzo****". Note currently Copy activity doesn't support LZO when read/write Parquet files*
* *XML format- The compression codec used to read/write XML files.  
  Allowed values are****bzip2****,****gzip****,****deflate****,****ZipDeflate***, **TarGzip**, **Tar**, **snappy**, or **lz4**

**When will you use a Parquet file over an ORC file?**

**Why would you choose an AVRO file format over a Parquet file form?**

*The biggest difference between ORC, Avro, and Parquet is how the store the data. Parquet and ORC both store data in columns, while Avro stores data in a row-based format. By their very nature, column-oriented data stores are optimized for read-heavy analytical workloads, while row-based databases are best for write-heavy transactional workloads.*

*Because of the way the data is optimized for fast retrieval, the column-based stores, Parquet and ORC, offer higher compression rates than the row-based Avro format,*

*Another aspect to consider is support for schema evolution, or the ability for the file structure to change over time. Among the two columnar formats, ORC offers better schema evolution*

*four key considerations: row or column, schema management, splitability, and compression.*