1. Why should one use Azure Key Vault when working in the Azure environment?

Azure Key Vault is a cloud service for securely storing and accessing secrets. A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, or cryptographic keys. Key Vault service supports two types of containers: vaults and managed hardware security module (HSM) pools. Vaults support storing software and HSM-backed keys, secrets, and certificates. Managed HSM pools only support HSM-backed keys.

1. What are the alternatives to using Azure Key Vault? What are the pros and cons of using Azure Key Vault?

**Pros:**

1. Data security is maximized because key vault provides a centralized location for your sensitive information.
2. Since it has a seamless integration with azure services and applications, key vault can be conveniently called in our data pipelines to retrieve the secrets.
3. We can maintain different versions of the password without the need to re-create a new line item in key vault.
4. Key vault is a scalable service.

**Cons:**

1. It’s not free.
2. A bit of a learning curve is involved.
3. Using key vault introduces dependency on Azure.
4. 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?

The ForEach Activity defines a repeating control flow in an Azure Data Factory or Synapse pipeline. This activity is used to iterate over a collection and executes specified activities in a loop.

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

Expressions in Azure Data Factory (ADF) are dynamic constructs written in Data Factory’s expression language. They are used to compute values at runtime — for example, to generate file names, filter data, assign parameters, or make decisions within a pipeline.

ADF expressions use functions, variables, parameters, and system variables and are typically written using @{} syntax.

Example:

@concat('input/file\_', formatDateTime(utcNow(), 'yyyyMMdd'), '.csv')

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

**Pros**

1. One dataset can serve multiple activities by changing its parameters (e.g., folder name, file name, table name).
2. Allows dynamic selection of source/sink paths or tables at runtime based on pipeline parameters.
3. Makes it easier to loop over different values
4. If a change is needed (e.g., file format), you update it in one place instead of editing multiple copies.

**Cons**

1. Complexity Increases
2. ADF can't fully validate parameterized datasets at design time, since values are resolved at runtime — this may lead to runtime errors if misconfigured.
3. Testing every possible combination of parameter values can be time-consuming.
4. 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?

Azure Data Factory supports the following file formats.

1. [Avro format](https://learn.microsoft.com/en-us/azure/data-factory/format-avro)
2. [Binary format](https://learn.microsoft.com/en-us/azure/data-factory/format-binary)
3. [Delimited text format](https://learn.microsoft.com/en-us/azure/data-factory/format-delimited-text)
4. [Excel format](https://learn.microsoft.com/en-us/azure/data-factory/format-excel)
5. [JSON format](https://learn.microsoft.com/en-us/azure/data-factory/format-json)
6. [ORC format](https://learn.microsoft.com/en-us/azure/data-factory/format-orc)
7. [Parquet format](https://learn.microsoft.com/en-us/azure/data-factory/format-parquet)
8. [XML format](https://learn.microsoft.com/en-us/azure/data-factory/format-xml)

**Parquet use cases:**

1. Storing big data of any kind (structured data tables, images, videos, documents).
2. When your full dataset has many columns, but you only need to access a subset.
3. When you want multiple services to consume the same data from object storage.

**Avro use cases:**

1. Write-heavy operations (such as ingestion into a data lake) due to serialized row-based storage.
2. When writing speed with schema evolution (adaptability to change in metadata) is critical.