Azure Mini Project: End-to-end ETL / Q&A

- **Q1** - Why should one use Azure Key Vault when working in the Azure environment? What are the pros and cons? What are the alternatives?

  A1 - Azure Key Vault provides a centralized solution for storing and managing keys, tokens, passwords and other secrets used within an application. Azure Key Vault allows for granular control over all types of access within applications while eliminating the need for developers to store security information within the code itself. It reduces code complexity while offering highly customizable security solutions. In addition to setting and storing key information, Azure Key Vault also allows for monitoring of access and use based on secrets. Additionally, Key Vault greatly simplifies management and administration of security information while seamlessly integrating with other Azure services.

  A potential drawback of using Azure Key Vault is that including it within your project does incur additional costs. There are two different pricing tiers, Standard (offering encryption based on software key) and Premium (with hardware security model protected keys). Though very affordable, these are considerations to take into account when budgeting for the overhead of a project. There are many alternatives to Azure Key Vault, such as Google Cloud Key Management or Amazon KMS.

- **Q2** - How do you achieve loop functionality within an Azure Data Factory pipeline? Why would you need to use this functionality in a data pipeline?

  A2 - One can achieve loop functionality within an Azure Data Factory pipeline by simply utilizing the ‘ForEach’ activity located within the ‘Activities’ panel under the ‘Iteration & conditionals’ subheading. This is a very useful feature which operates similar to any for-loop would within most programming languages. It allows you to iterate over a collection and execute an action on each item within that collection. An example of a common use case for utilizing a ForEach action within a pipeline would be when needing to copy all elements from a directory to move them into a different directory, or moving files from local storage to cloud storage. The ForEach activity also makes it easy to run pipelines within pipelines.

- **Q3** - What are expressions in Azure Data Factory? How are they helpful when designing a data pipeline? Please explain with an example.

  A3 - Expressions in Data Factory are a powerful tool allowing you to transform, filter, and apply functions and parameters to data within your dataset and pipeline. They allow you to slice and dice your dataset and perform the necessary transformations upon said data. An example would be when parsing a date from a file or column name as a string to output as a timestamp. With Expression Builder you can easily convert the string representation of a date to a timestamp by passing it to the 'toTimestamp' function.

- **Q4** - What are the pros and cons of parametrizing a dataset's activity in Azure Data Factory?

  A4 - Parameterization of datasets within Azure Data Factory saves you time, allows for more flexible pipeline solutions, and allows you to better control flow within your data pipeline. Parameters upon datasets save you from having to create additional objects within Data Factory, instead you can pass dynamic variables to the dataset, or even parameterize pipelines to further simplify your processes.

- **Q5** - 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?

  A5 - Azure Data Factory supports the following file formats: Avro, Binary, Delimited text (CSV, TSV, etc), Excel, JSON, ORC, Parquet, and XML.
  Note some complex data types are not supported within Data Factory Copy activity.

  Compression Codecs by file type:
  Avro - "none" (default), "deflate", "snappy"
  Binary - bzip2, gzip, deflate, ZipDeflate, Tar, or TarGzip.
  Delimited text - bzip2, gzip, deflate, ZipDeflate, TarGzip, Tar, snappy, or lz4.
  Excel (out as JSON) - bzip2, gzip, deflate, ZipDeflate, TarGzip, Tar, snappy, or lz4.
  JSON - bzip2, gzip, deflate, ZipDeflate, TarGzip, Tar, snappy, or lz4.
  ORC - none, zlib, snappy (default), and lzo.
  Parquet - none", "gzip", "snappy" (default), and "lzo".

You would likely want to choose Parquet format over ORC format if you are working with heavily nested data or for an analytics workload. It more efficiently handles data IO operations. It is the default file format for reading / writing data in Apache Spark and offers a better range of support within the Hadoop ecosystem.

Avro is a row-based storage format while Parquet is a columnar format. As such, if you are working with row formatted data, Avro may be a better solution. Avro allows for better performance within write-heavy workloads and offers much more flexible schema evolution such as modifying or adding columns. Avro is better suited to ETL operations in scenarios where you need to query all of the columns.