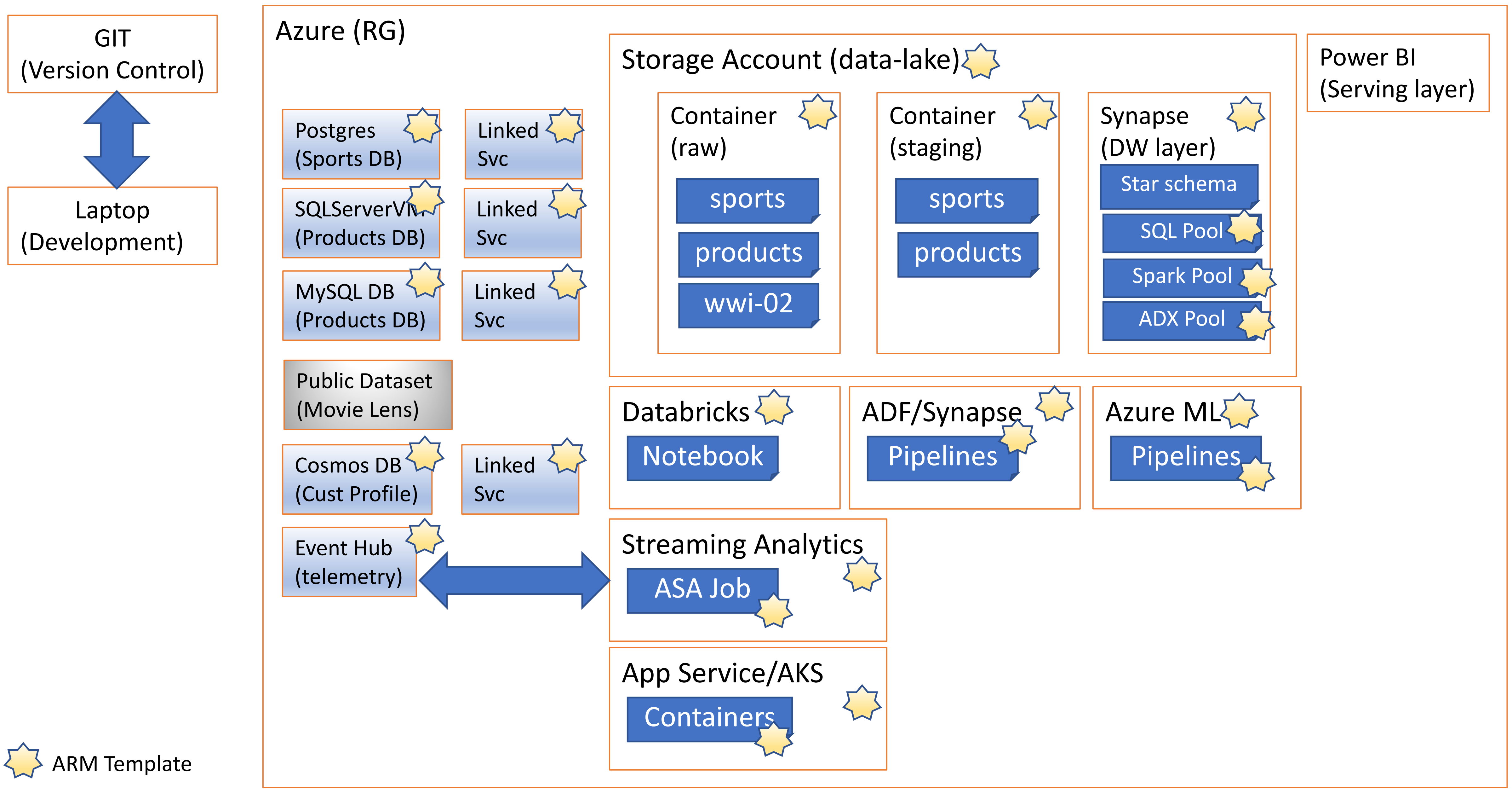
DP-203 Notes

Sample Project Architecture



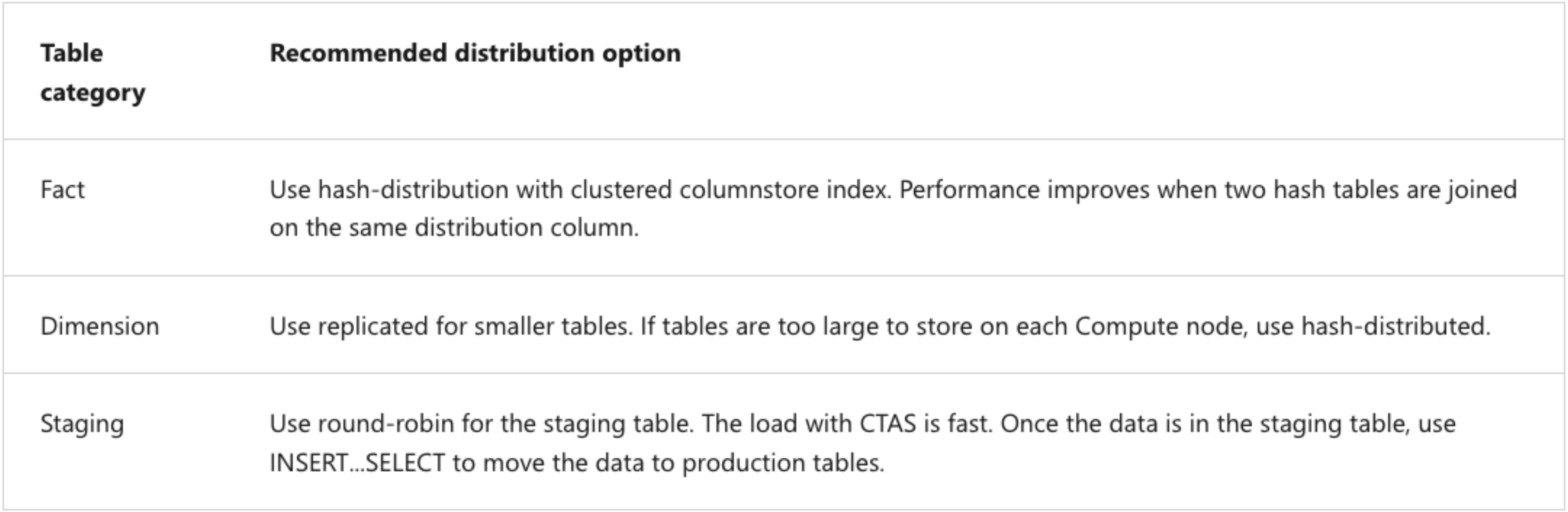
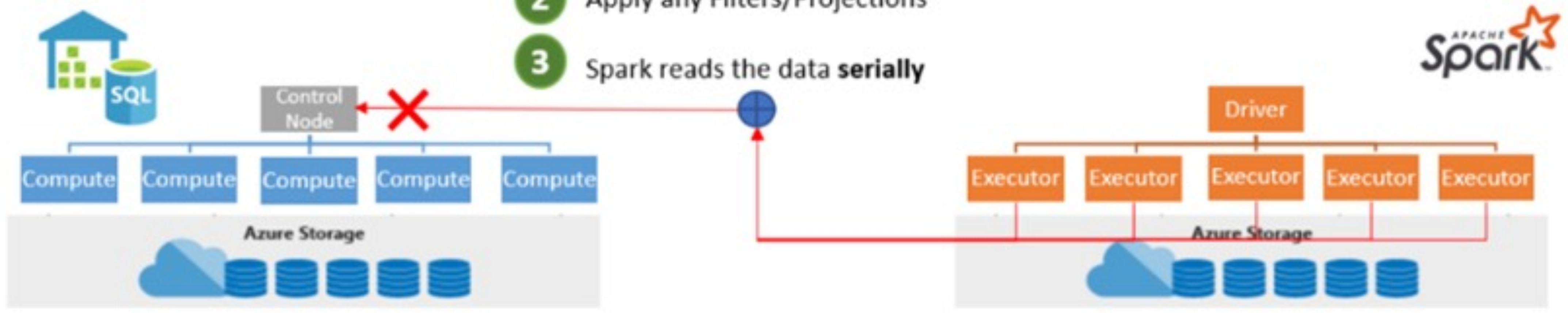


Table category	Recommended distribution option
Fact	Use hash-distribution with clustered columnstore index. Performance improves when two hash tables are joined on the same distribution column.
Dimension	Use replicated for smaller tables. If tables are too large to store on each Compute node, use hash-distributed.
Staging	Use round-robin for the staging table. The load with CTAS is fast. Once the data is in the staging table, use INSERT...SELECT to move the data to production tables.

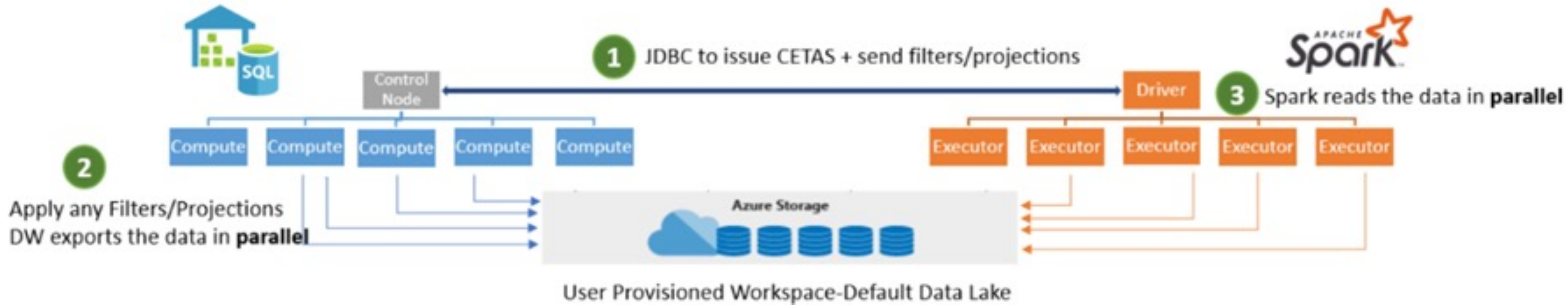
https://docs.microsoft.com/en-us/learn/modules/optimize-data-warehouse-query-performance-azure-synapse-analytics/5-use-indexes-to-improve
https://docs.microsoft.com/en-us/learn/modules/understand-data-warehouse-developer-features-of-azure-synapse-analytics/3-understand-transact-sql-language-capabilities
https://docs.microsoft.com/en-us/learn/modules/query-data-lake-using-azure-synapse-serverless-sql-pools/4-external-objects

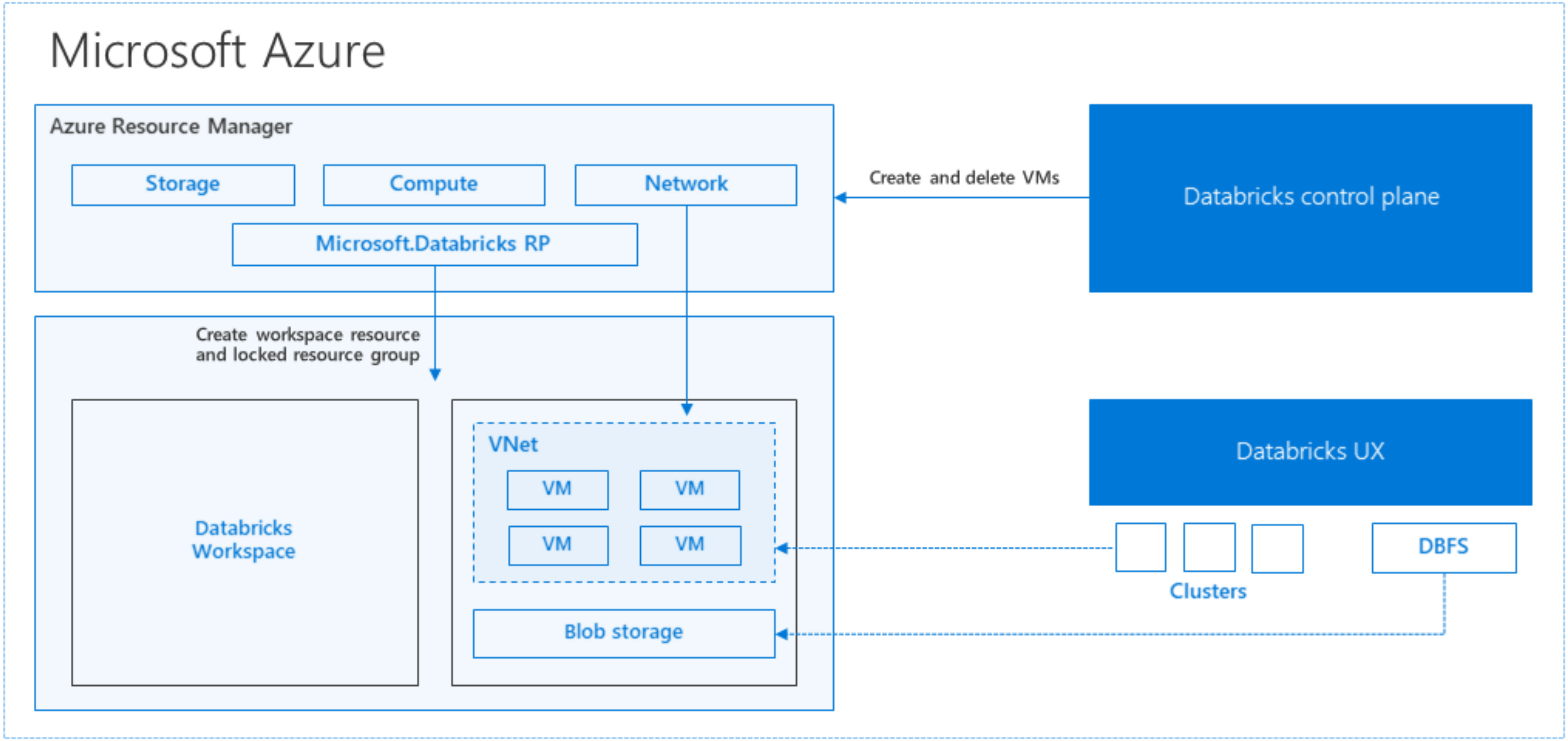
Existing Approach: JDBC

1. JDBC to open connection
2. Apply any Filters/Projections
3. Spark reads the data **serially**



New Approach: JDBC and Polybase


1 JDBC to issue CETAS + send filters/projections
2 Apply any Filters/Projections
DW exports the data in parallel
3 Spark reads the data in parallel
SQL
Control Node
Compute
Compute
Compute
Compute
Compute
Apache Spark
Driver
Executor
Executor
Executor
Executor
Executor
Azure Storage
User Provisioned Workspace-Default Data Lake


Microsoft Azure
Azure Resource Manager
Storage
Compute
Network
Microsoft.Databricks RP
Create and delete VMs
Databricks control plane
Create workspace resource and locked resource group
Databricks Workspace
VNet
VM
VM
VM
VM
Blob storage
Databricks UX
Clusters
DBFS

```sql
SELECT doc
FROM
    OPENROWSET(
        BULK 'https://mydatalake.blob.core.windows.net/data/files/*.json',
        FORMAT = 'csv',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    ) WITH (doc NVARCHAR(MAX)) as rows
```

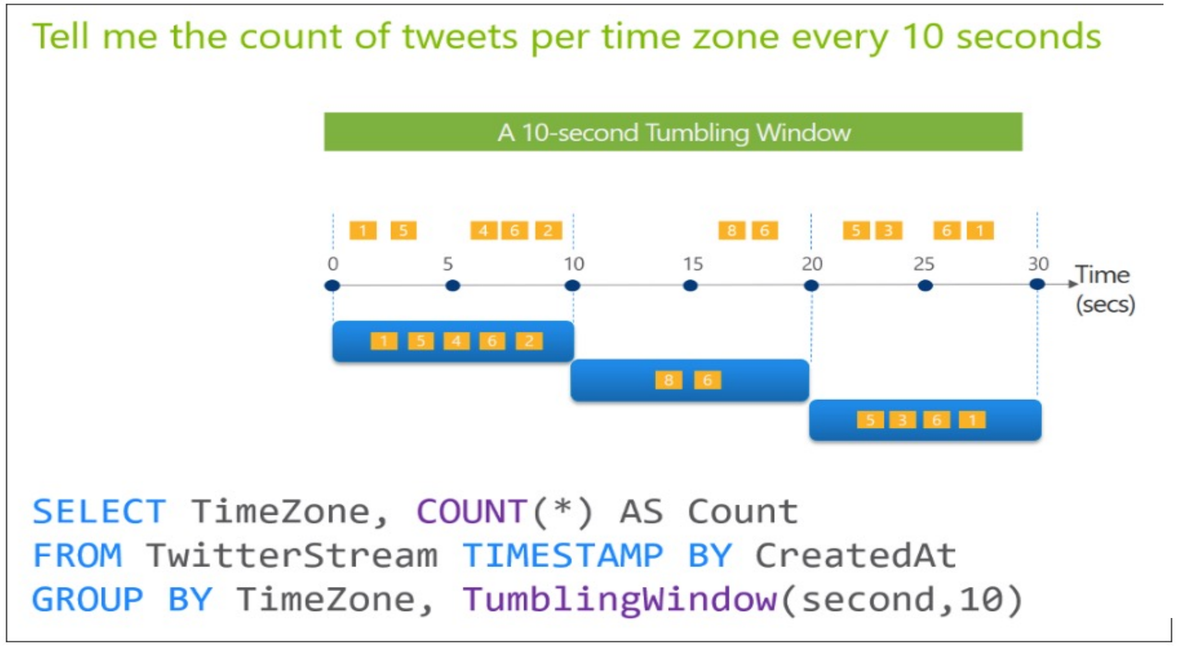
OPENROWSET has no specific format for JSON files, so you must use *csv* format with **FIELDTERMINATOR**, **FIELDQUOTE**, and **ROWTERMINATOR** set to *0x0b*, and a schema that includes a single NVARCHAR(MAX) column. The result of this query is a rowset containing a single column of JSON documents, like this:

SQL   Copy

```sql
SELECT *
FROM OPENROWSET(
    BULK 'https://mydatalake.blob.core.windows.net/data/orders/year=*/month=*/*.*',
    FORMAT = 'parquet') AS orders
WHERE orders.filepath(1) = '2020'
    AND orders.filepath(2) IN ('1','2');
```

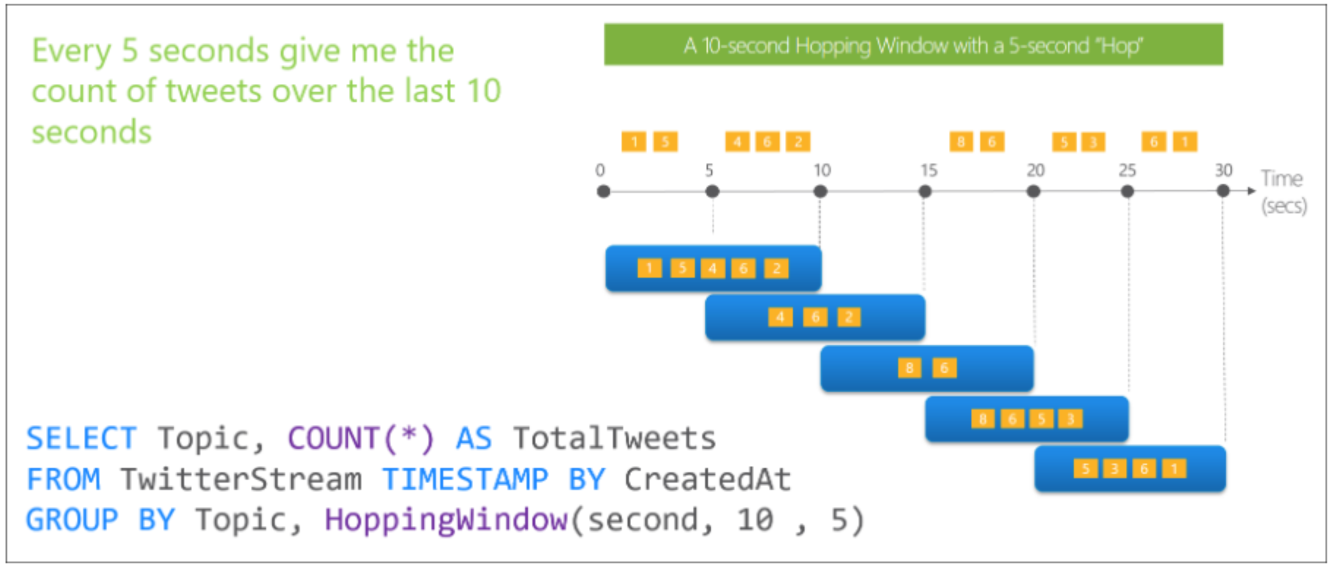
The numbered filepath parameters in the WHERE clause reference the wildcards in the folder names in the BULK path -so the parameter 1 is the * in the *year=** folder name, and parameter 2 is the * in the *month=** folder name.

Tumbling windows are a series of fixed-sized, non-overlapping and contiguous time intervals. The following diagram illustrates a stream with a series of events and how they are mapped into 10-second tumbling windows.


Tell me the count of tweets per time zone every 10 seconds
A 10-second Tumbling Window
1 5 4 6 2 8 6 5 3 6 1
0 5 10 15 20 25 30
Time (secs)

```
SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
```

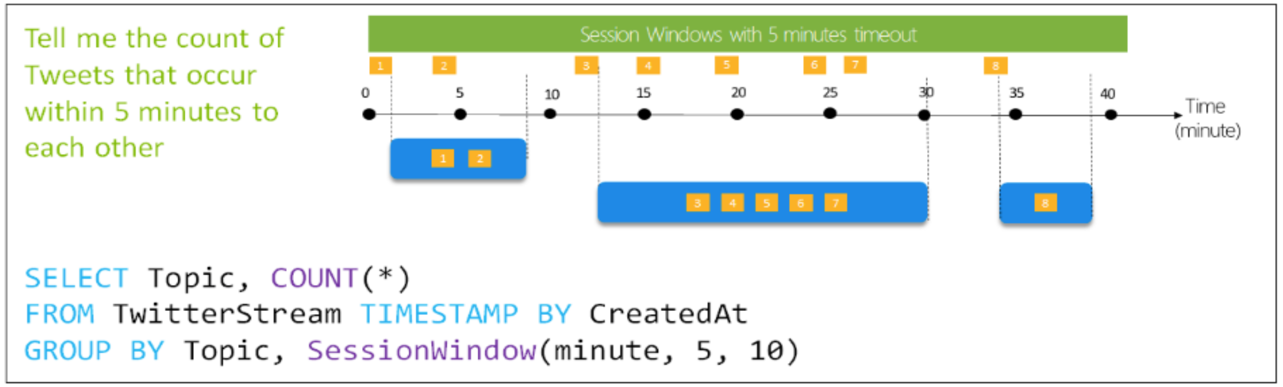
The following illustration shows a stream with a series of events. Each box represents a hopping window and the events that are counted as part of that window, assuming that the 'hop' is 5, and the 'size' is 10.


Every 5 seconds give me the count of tweets over the last 10 seconds
A 10-second Hopping Window with a 5-second "Hop"
1 5 4 6 2 8 6 5 3 6 1
0 5 10 15 20 25 30
Time (secs)

```
SELECT Topic, COUNT(*) AS TotalTweets
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, HoppingWindow(second, 10 , 5)
```

Session windows group events that arrive at similar times, filtering out periods of time where there is no data. Session window function has three main parameters: timeout, maximum duration, and partitioning key (optional).

The following diagram illustrates a stream with a series of events and how they are mapped into session windows of 5 minutes timeout, and maximum duration of 10 minutes.


Tell me the count of Tweets that occur within 5 minutes to each other
Session Windows with 5 minutes timeout
1 2 3 4 5 6 7 8
0 5 10 15 20 25 30 35 40
Time (minute)

```
SELECT Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, SessionWindow(minute, 5, 10)
```

You should use the session windowing function to group events that arrive at a similar time and filter out time periods where no data exists. Windowing functions are native to Stream Analytics, which is what you should use to analyze the data. The session windowing function allows you to group streaming events that arrive at a similar time and filter out time periods where no data exists.

You should use the tumbling windowing function to count the number of events that are received per time zone every minute. This function allows you to segment data into distinct time segments. A tumbling windowing function is then applied to the data in each segment. An example is as follows:

```
SELECT TimeZone, Count(*) as Count
FROM WeatherStream TIMESTAMP BY ReportedAt
GROUP BY TimeZone, TumblingWindow(minute, 1)
```

You should use the sliding windowing function to count the number of events received in a time zone during the last 30 seconds. This function produces output only when an event occurs. You can use the following query:

```
SELECT TimeZone, Count(*)
FROM WeatherStream TIMESTAMP BY ReportedAt
GROUP BY TimeZone, SlidingWindow(second, 30)
```

You should use the hopping windowing function to retrieve the number of events every five seconds by time zone during the last 30 seconds. A hopping window function looks backwards to determine when an event occurs. Events can overlap. You can use the following query:

```
SELECT TimeZone, Count(*)
FROM WeatherStream TIMESTAMP BY ReportedAt
GROUP BY TimeZone, HoppingWindow(second, 30, 5)
```

Tumbling window functions are used to segment a data stream into distinct time segments and perform a function against them, such as the example below. The key differentiators of a Tumbling window are that they repeat, do not overlap, and an event cannot belong to more than one tumbling window.

A: Sliding windows, unlike Tumbling or Hopping windows, output events only for points in time when the content of the window actually changes. In other words, when an event enters or exits the window. Every window has at least one event, like in the case of Hopping windows, events can belong to more than one sliding window.

C: Hopping window functions hop forward in time by a fixed period. It may be easy to think of them as Tumbling windows that can overlap, so events can belong to more than one Hopping window result set. To make a Hopping window the same as a Tumbling window, specify the hop size to be the same as the window size.

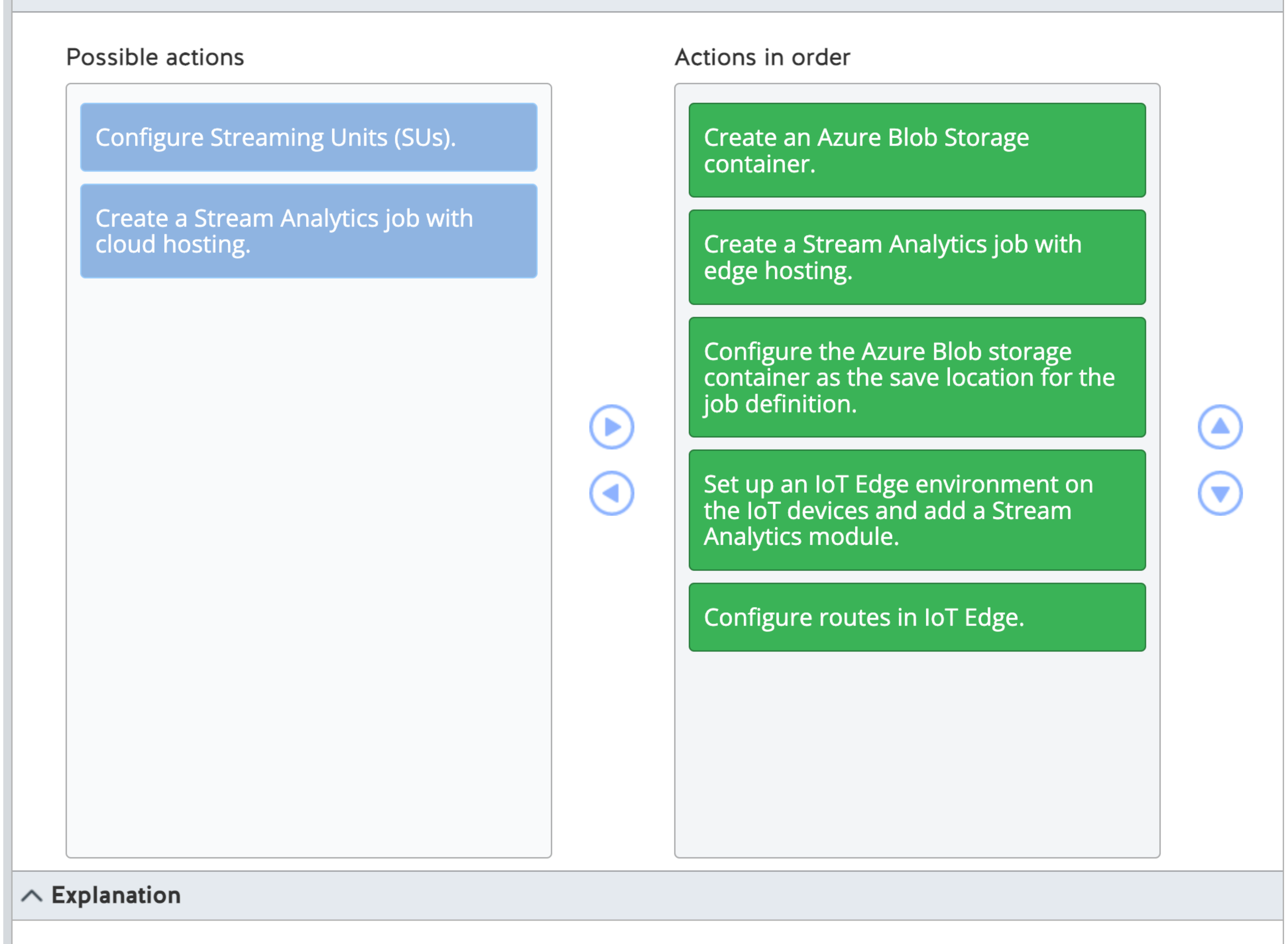
D: Session windows group events that arrive at similar times, filtering out periods of time where there is no data.

Watermark Delay indicates the delay of the streaming data processing job.
There are a number of resource constraints that can cause the streaming pipeline to slow down. The watermark delay metric can rise due to:
1. Not enough processing resources in Stream Analytics to handle the volume of input events. To scale up resources, see Understand and adjust Streaming Units.
2. Not enough throughput within the input event brokers, so they are throttled. For possible solutions, see Automatically scale up Azure Event Hubs throughput units.
3. Output sinks are not provisioned with enough capacity, so they are throttled. The possible solutions vary widely based on the flavor of output service being used.
https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-time-handling

Information classification : https://docs.microsoft.com/en-us/azure/azure-sql/database/data-discovery-and-classification-overview

ADLS Security : https://docs.microsoft.com/en-us/azure/data-lake-store/data-lake-store-secure-data

https://azure.microsoft.com/en-in/blog/maximize-throughput-with-repartitioning-in-azure-stream-analytics/
https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-streaming-unit-consumption



Possible actions

Configure Streaming Units (SUs).

Create a Stream Analytics job with cloud hosting.

Actions in order

Create an Azure Blob Storage container.

Create a Stream Analytics job with edge hosting.

Configure the Azure Blob storage container as the save location for the job definition.

Set up an IoT Edge environment on the IoT devices and add a Stream Analytics module.

Configure routes in IoT Edge.

Explanation