Develop data processing (40-45%):

DP-203 exam – Skills Measured 2

Develop data processing

2.1 Ingest and transform data
2.2 Develop a batch processing solution
2.3 Develop a stream processing solution
2.4 Manage batches and pipelines

Transform data by using Transact-SQL (T-SQL)

T-SQL runs INSERT, UPDATE, and DELETE operations on a target table from the results of a join with a source table.

Use Merge to perform:

- INSERT and UPDATE operations on a table in a single statement
- UPDATE and DELETE operations on a table in a single statement
- UPDATE and INSERT operations on a target table by using a derived source table
- INSERT or UPDATE on a target edge table in a graph database
- Insert the results of the MERGE statement into another table

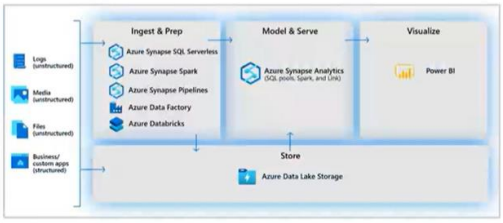
Some of the key activities to transform data are as follows:

Activity	Capabilities
Script activity	Allows users to insert, modify, delete and retrieve data in the database and create, modify, and remove database objects, such as tables, indexes, and users.
Stored procedure activity	Enables users to invoke a stored procedure in Azure SQL Database, Azure Synapse Analytics, or SQL Server Database.

Ingest and transform data by using Azure Synapse Pipelines or ADF

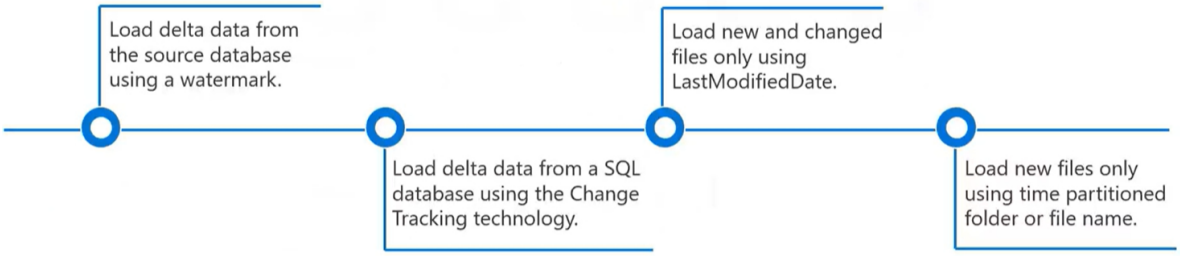
Perform data transformations with Azure Synapse pipelines code free using the Mapping Data Flow task.

- Ingest data using ADF as it empowers you to do code-free Extract, Transform, Load (ETL) or Extract, Load, Transform (ELT), including preparation and transformation.
- Consider a hybrid approach and combine ADF, Databricks and Power BI with Azure Synapse Analytics.



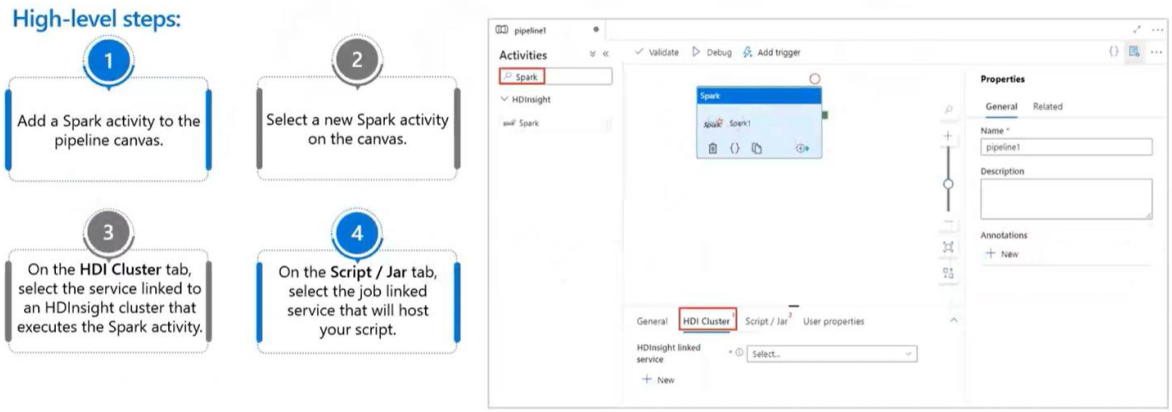
Design and implement incremental loads

There are different techniques to implement incremental loads using Azure Data Factory (ADF):



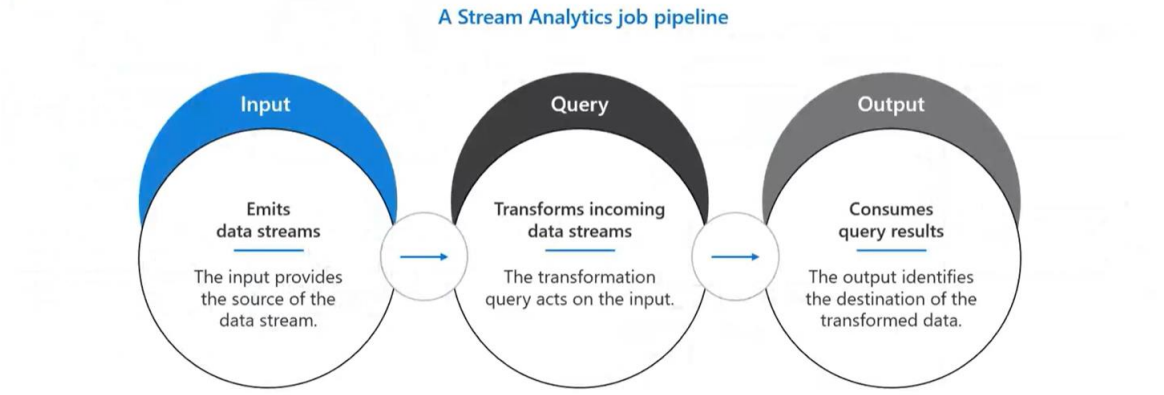
Transform data by using Apache Spark

You can transform data using Spark activity in ADF and Synapse Analytics.



Transform data by using Azure Stream Analytics

The Stream Analytics pipeline provides a transformed data flow from input to output.

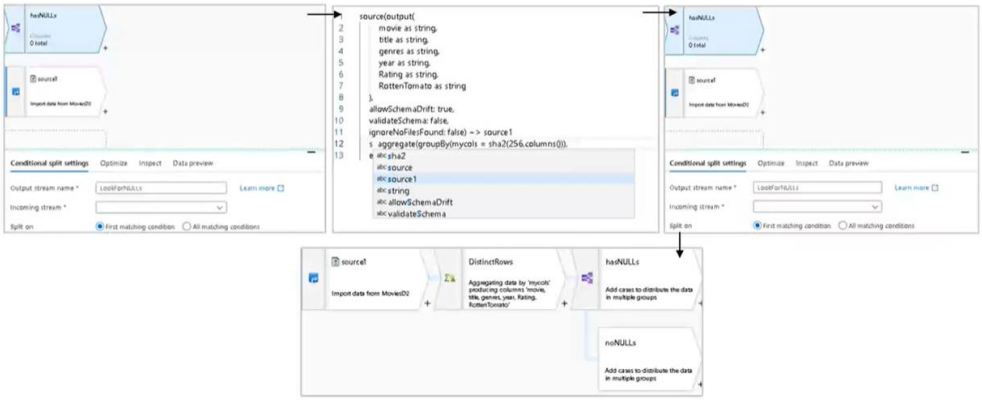


Cleanse data

The data ingested from source locations is cleansed, normalized, and processed for other tasks using Apache Spark in Azure Synapse Analytics and Databricks.

Handle duplicate data

You can easily perform tasks such as data deduplication and null filtering by using code snippets in mapping data flows.



Handle missing data

Azure Databricks provides a unified interface for handling bad records and files without interrupting Spark jobs.

Errors

- File not readable due to the file being missing, inaccessible, or corrupted
- Records are not parsable due to syntax error or schema mismatch

Example 1: Unable to find input file

```
Scala

val df = spark.read
  .option("badRecordsPath", "/tmp/badRecordsPath")
  .parquet("/input/parquetFile")

// Delete the input parquet file '/input/parquetFile'
dbutils.fs.rm("/input/parquetFile")

df.show()
```

Resolution

- Set the data source option **badRecordsPath** to specify a path to store exception files
- Ignore transient errors like network connection exception, IO exception, and so on, and record under the **badRecordsPath**

Example 2: Input file contains bad record

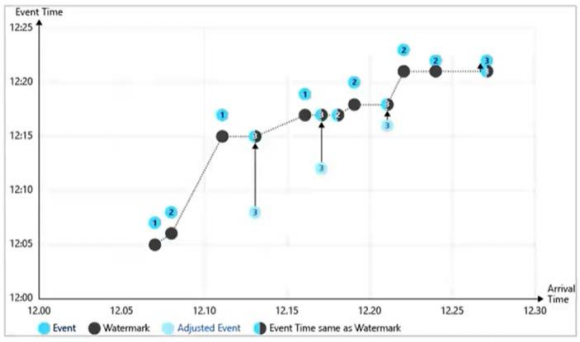
```
// Creates a json file containing both parsable and corrupted records
Seq("""{"a": 1, "b": 2}""", """{bad-record""").toDF().write.text("/tmp/input/jsonFile")

val df = spark.read
  .option("badRecordsPath", "/tmp/badRecordsPath")
  .schema("a int, b int")
  .json("/tmp/input/jsonFile")

df.show()
```

Handle late-arriving data

For each incoming event, Azure Stream Analytics compares the event time with the arrival time.

Features

- If the event time is outside of the tolerance window, configure the system to drop the event or adjust the event's time to be within the tolerance.
- After watermarks are generated, the service can potentially receive events with an event time lower than the watermark.
- As a part of the adjustment, the event's **System.Timestamp** is set to the new value, but the event time field itself is not changed.

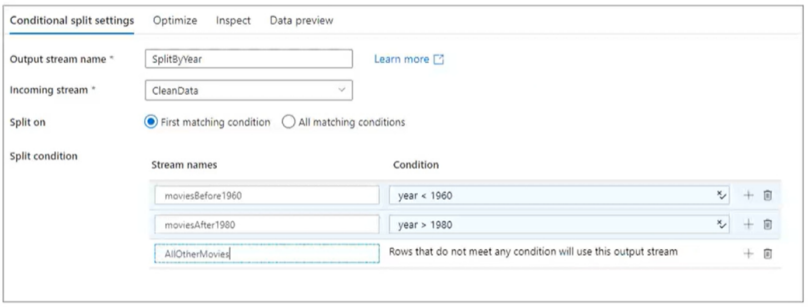


Split data

The conditional split transformation:

- Routes data rows to different streams based on matching conditions.
- Is similar to a CASE decision structure in a programming language.
- Evaluates expressions and directs the data row to the specified destination based on the results.

With the **Split on** setting, you can determine whether the row of data flows to the first matching stream or every matching stream.



Shred JSON

Read JSON Documents

You can read the JSON documents in the following ways:

- Provide the file URL to the **OPENROWSET** function.
- Use OPENJSON(BULK) that reads the content of the file and returns it in BulkColumn.

Parse JSON Documents

You can parse the values in the JSON documents and return them as relational values in the following ways:

- Use function **JSON_VALUE**
- Use function **OPENJSON**

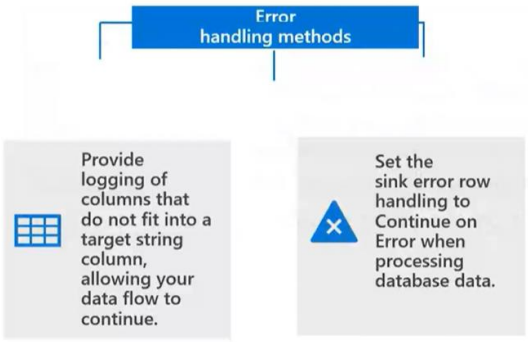
Encode and decode data

These conversion functions (supported by ADF and Azure Synapse Analytics) in mapping data flows help to code and encode data.

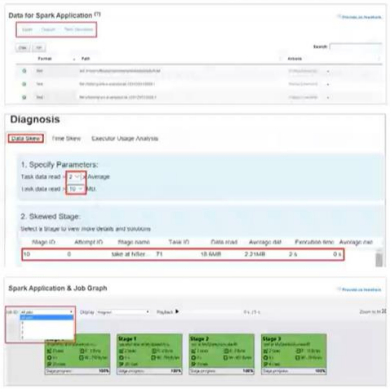
Conversion function	Task
ascii	It returns the numeric value of the input character. For the input strings that have more than one character, the numeric value of the first character is returned.
char	It returns the ascii character represented by the input number. If number is greater than 256, the result is equivalent to char(number % 256).
decode	It decodes the encoded input data into a string based on a given charset.
encode	It encodes the input string data into binary based on a charset.

Configure error handling for a transformation

A common error condition that you must prevent is possible column truncation.



The Extended Apache Spark history server helps debug and diagnose completed and running Spark applications.



Normalize and denormalize values

Normalization is organizing a database to reduce redundancy and improve data integrity.

PO	Vendor	Apple	Pear
#1	A	1	3
#2	B	2	4

PO	Vendor	Fruit	Amount
#1	A	Apple	1
#1	A	Pear	3
#2	B	Apple	2
#2	B	Pear	4

Denormalization is adding redundant data to your schema to eliminate joins when querying. For example, **Flatten**.

A	B	C	D	E	F
			1		
			2		

A	B	C	1_e	1_f	2_e	2_f

Perform exploratory data analysis

There are three steps in the Exploratory Data Analysis (EDA) process in Databricks:

1 Data Acquisition

- Collect or obtain the data from various sources.
- Check the format of the data.
- Check the quality of the data.

2 Data Exploration

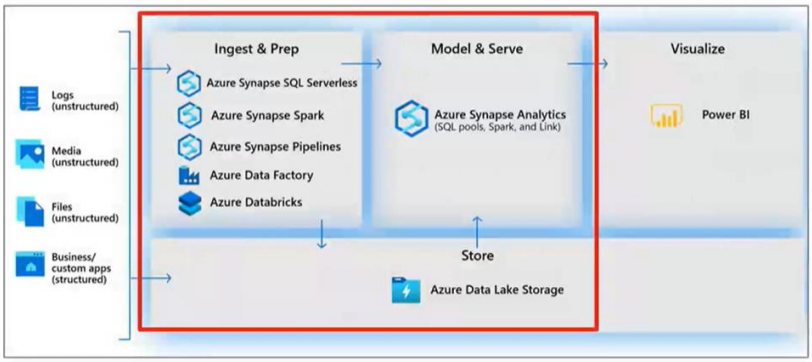
- Explore and visualize the data.
- Identify and handle issues and anomalies.
- Transform and manipulate data to create new features or variables.

3 Data Evaluation

- Evaluate and interpret the results of the data exploration.
- Communicate and present the data..

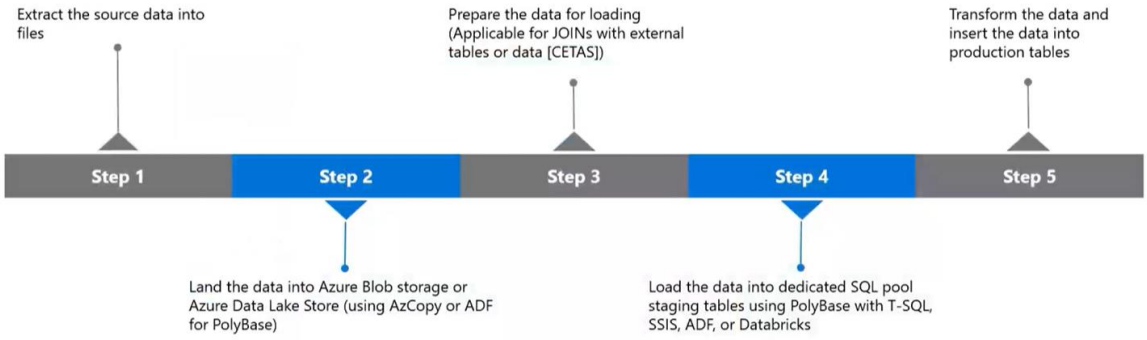
Develop batch processing solutions by using Azure Data Lake Storage, Azure Databricks, Azure Synapse Analytics, and ADF

A modern data warehouse enables you to collate all your data at scale easily, so you get to the insights through analytics dashboards, operational reporting, or advanced analytics for your users.



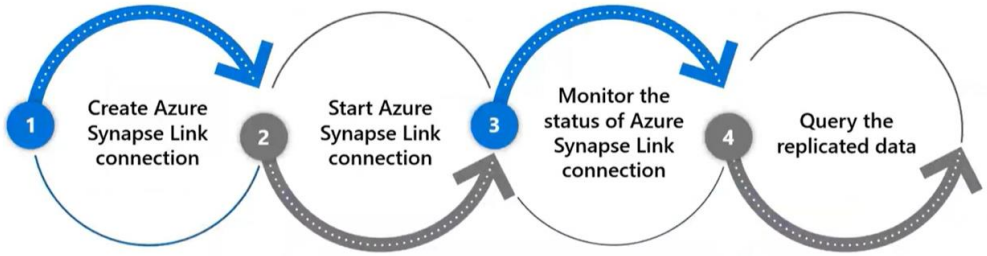
Use PolyBase to load data to a SQL pool

Remember key aspects to load data using PolyBase with these basic steps for a PolyBase Extract, Load, and Transform (ELT).



Implement Azure Synapse Link and query the replicated data

Some of the key steps to create the Azure Synapse Link for Azure SQL Database are as follows:

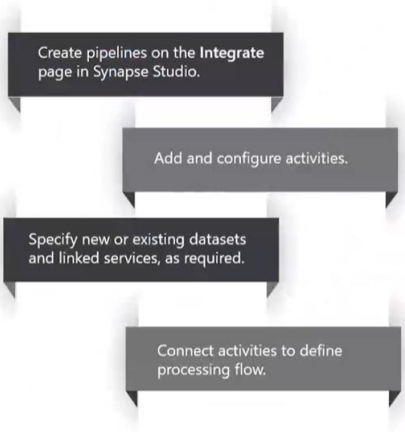


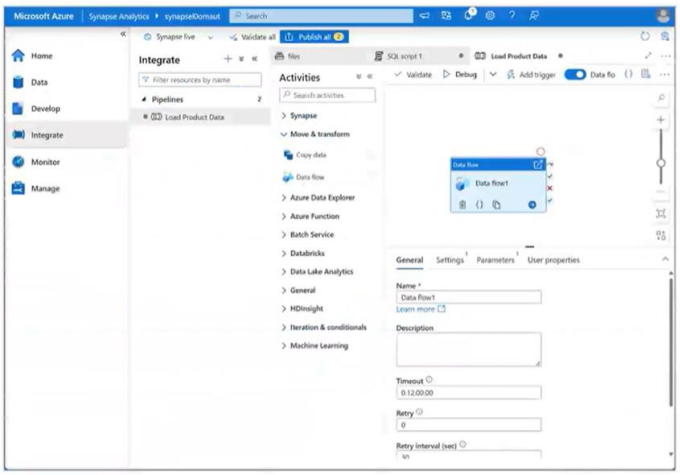
Note:

1. You can also use Azure Synapse Link for Azure Cosmos DB that enables near real-time analytics over operational data in Azure Cosmos DB.
2. However, remember that the key difference is that the analytical store is maintained in the Synapse dedicated SQL pool for Azure SQL database, but maintained in Cosmos database when using the same.

Create data pipelines

Remember these key steps to create pipelines.

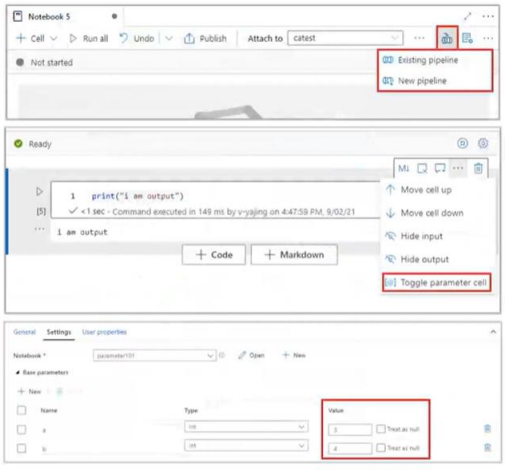




Integrate Jupyter notebooks into a Synapse data pipeline

Integrate a Jupyter or a Python notebook into a data pipeline as follows:

1. Add a notebook to a new or existing pipeline.
2. Designate the cells as the parameter cells.
3. Assign parameter values.



Upsert data in batch processing

There are two major approaches to upsert data in batch processing.

For smaller number of records:

- Iterate on the dataset on the application tier.
- For every row, invoke a stored procedure to execute an INSERT/UPDATE operation depending on the existence of record with a certain key.

For large number of records:

- Leverage bulk insert techniques to upload the entire dataset to Azure SQL Database.
- Execute all the INSERT/UPDATE (or MERGE) operations within a single batch to minimize roundtrips and log writes and maximize throughput.

You can use UPDATE and INSERT statements to implement Type 1 and 2 dimensions, or single MERGE statement to perform an "upsert" operation to insert new records and update existing ones.

Configure batch retention and revert data to a previous state

Use Dedicated SQL Pool restore points to recover or copy the data warehouse to a previous state in the primary region.

Data warehouse snapshot creates a restore point to recover or copy the data warehouse to a previous state.

Data warehouse restore is a new data warehouse created from a restore point of an existing or deleted data warehouse.

Automatic Restore Points

- Snapshots are a built-in feature that creates restore points.
- Dedicated SQL Pool should be in active state

User-defined Restore Points

- Manually trigger snapshots to create restore points of the data warehouse before and after large modifications.
- Available for seven days and are automatically deleted.

Reverting/restoring data: Each snapshot creates a restore point that represents the time the snapshot started. To restore a data warehouse, choose a restore point and issue a restore command.

Read from and write to a Delta Lake

Delta Lakes are used to store data and tables.

Reading from a Delta Lake

Access the data in the Delta Lakes by:

- Specifying the path to the files: "/tmp/delta-table" (with Apache Spark Cluster)
- Leveraging the well-defined protocol of the Delta Lake transaction log.

Writing to a Delta Lake

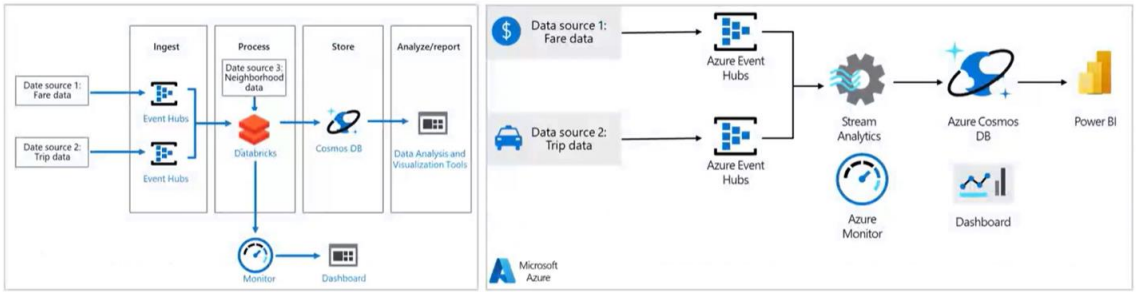
Remember the standard syntax for writing data in the Delta Lake by.

- Using the append mode to atomically add new data to an existing Delta table.
- Using the overwrite mode to atomically replace all the data in a table.

Delta Lake also brings atomicity, consistency, isolation, and durability (ACID) transactions to Apache Synapse Spark Pools and Databricks.

Create a stream processing solution

The reference architectures below show end-to-end stream processing pipelines with Databricks and Stream Analytics.

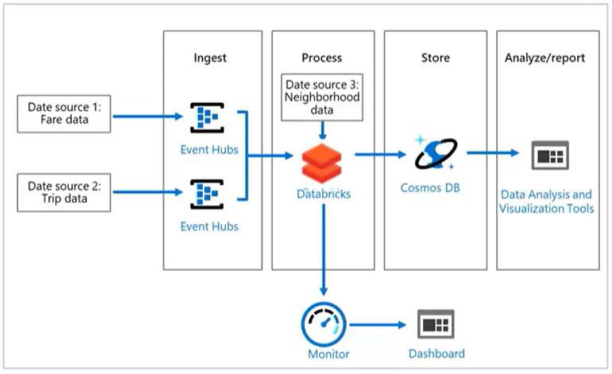


Process data by using Spark structured streaming

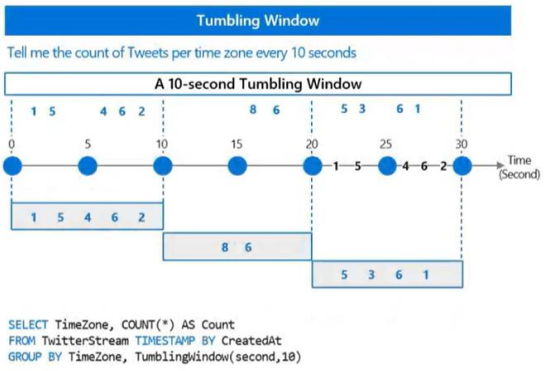
Apache Spark jobs are triggered by the Azure Synapse connector to read data from and write data to the Blob storage container.

The Azure Synapse connector uses three types of network connections:

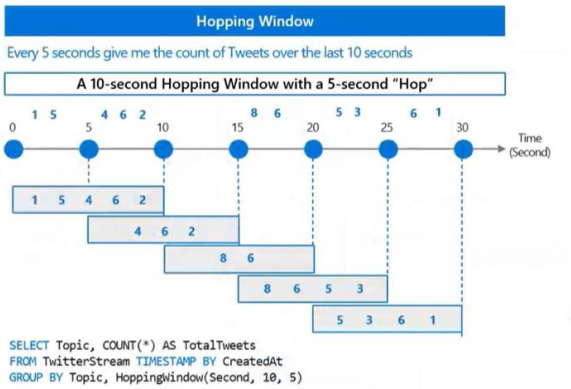
- Spark driver to Azure Synapse
- Spark driver and executors to Azure storage account
- Azure Synapse to Azure storage account



Create windowed aggregates (1/2)

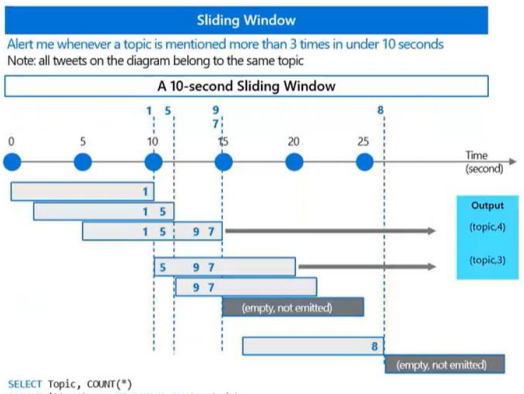



```
SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
```

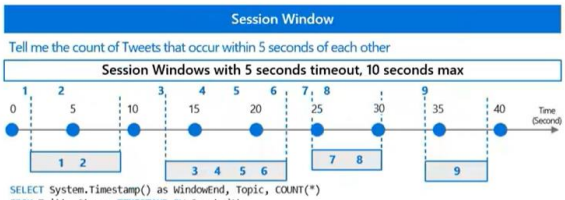



```
SELECT Topic, COUNT(*) AS TotalTweets
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, HoppingWindow(Second, 10, 5)
```

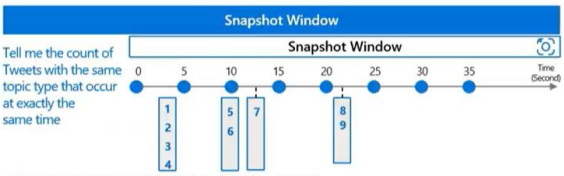
Create windowed aggregates (2/2)




```
SELECT Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, SlidingWindow(second, 10)
HAVING COUNT(*) >= 3
```




```
SELECT System.Timestamp() as WindowEnd, Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, SessionWindow(Second, 5, 10)
```



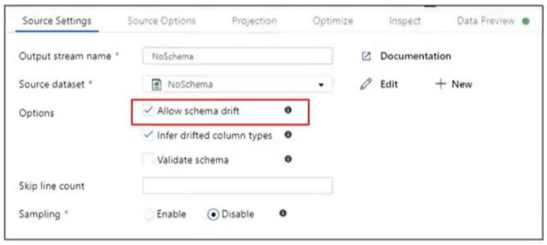

```
SELECT System.Timestamp() as WindowEnd, Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, System.Timestamp()
```

Handle schema drift

Schema drift is the case where your sources often change metadata. There can be two types of schema drift handling:

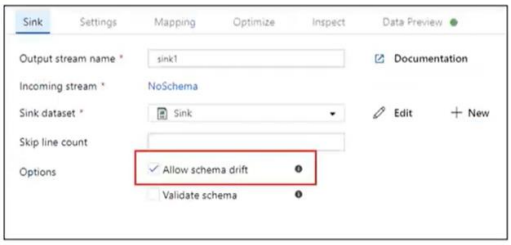
Schema drift in source

Columns coming into your data flow from your source definition are not present in your source projection.



Schema drift in sink

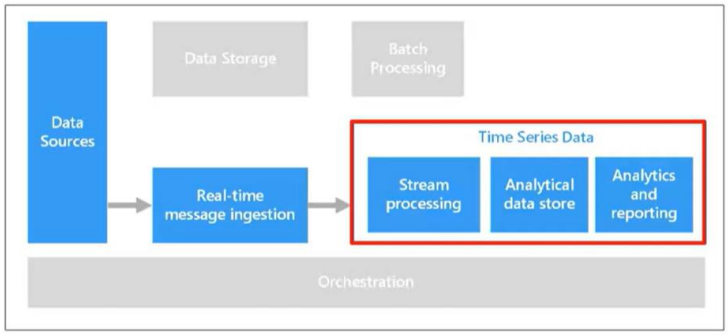
Additional columns are written on top of what is defined in the sink data schema.



Process time series data

Time series-based systems, such as Internet of things (IoT) scenarios, capture data in real time by using a real-time processing architecture.

You can use a Stream Analytics *job* to collect data from an event hub and send it to your Azure Data Explorer cluster.



Process data across partitions and within one partition

Use repartitioning and parallelization to optimize pipelines.

Parallelization

(Process data across partition)

Stream Analytics job can consume and write different partitions in parallel, which increases throughput.

An embarrassingly parallel job connects one partition of the input to one instance of the query to one partition of the output. Few examples include:

- Eight event hub input partitions and eight event hub output partitions.
- Eight event hub input partitions and blob storage output.

Repartitioning

(Process data within partition)

Required when you process data on a stream that's not sharded according to a natural input scheme.

Repartition can be added in two ways:

- Use a separate Stream Analytics job that does the repartitioning.
- Use a single job but do the repartitioning before your custom analytics logic.

Configure watermarking during processing

With Structured Streaming, you can use watermarks to control the threshold for how long to continue processing updates for a given state entity (such as aggregations over a time window or unique keys in a join between two streams).

Example 1: Applies a 10-minute watermark threshold to a windowed count:

```
from pyspark.sql.functions import window (df
.withWatermark("event_time", "10 minutes") .groupBy(
window("event_time", "5 minutes"), "id") .count() )
```

Example 2: An example query with stream-stream joins.

```
val inputStream1 = ... // delays up to 1 hour val inputStream2 = ... // delays up to 2 hours
inputStream1.withWatermark("eventTime1", "1 hour") .join(
inputStream2.withWatermark("eventTime2", "2 hours"), joinCondition)
```

Example 3: Use the **dropDuplicatesWithinWatermark** method:

```
streamingDf = spark.readStream. ...
# deduplicate using guid column with watermark based on eventTime column (streamingDf
.withWatermark("eventTime", "10 hours")
.dropDuplicatesWithinWatermark("guid") )
```

Note: Azure Stream Analytics generates heuristic watermarks based on the incoming events. We can observe several Event ordering time tolerance effects through Azure Stream Analytics job metrics like watermark delay, late input events, and so on.

Scale resources in stream processing

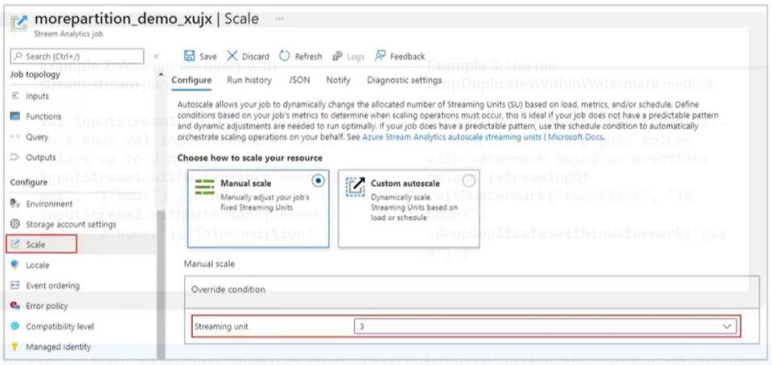
To scale resources, ensure you:

Understand and adjust streaming units:

Select the manual or custom auto method to scale and set the number of streaming units

Create parallelized jobs:

Leverage partitions in the input and output of a Stream Analytics Job to scale



Create tests for data pipelines in batch and stream processing

The local testing options supported by Azure Stream Analytics tools are:

Input	Output	Job Type
Local static data	Local static data	Cloud/Edge
Live input data	Local static data	Cloud
Live input data	Live output data	Cloud

You can create tests for data pipelines by:

1. Running queries on automatically sampled incoming data.
2. Running queries on sample data uploaded from a local file.
3. Troubleshooting errors in input and query size.

Optimize pipelines for analytical or transactional purposes

You can use the following methods to optimize processing with Azure Stream Analytics:

Repartitioning

It allows each shard to be processed independently to linearly scale out streaming pipeline.

Parallelization

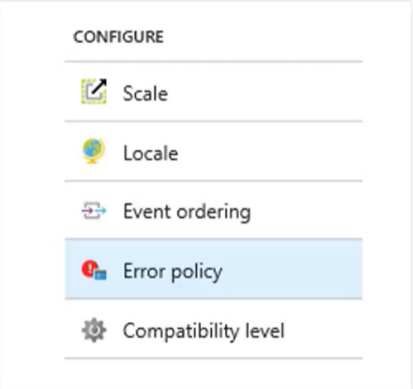
A Stream Analytics job consumes and writes different partitions in parallel to increase throughput.

Handle interruptions

Output error policy applies to errors when the output event produced by a Stream Analytics job does not conform to the schema of the target sink.

Consider the following configurations to handle interruptions:

- **Retry**
  - Retries writing the event indefinitely until the write succeeds.
  - Blocks all subsequent events from processing by the event that is retrying.
- **Drop**
  - Drops any output event that results in a data conversion error.



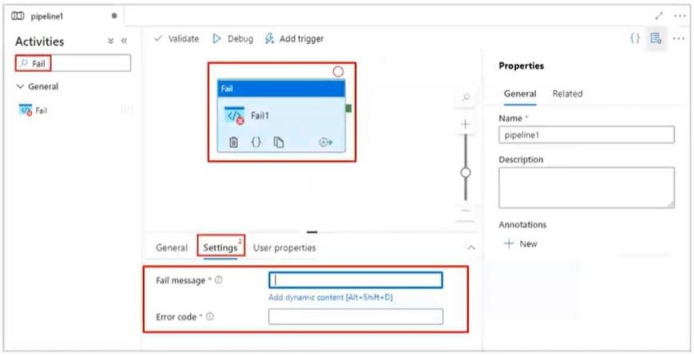
Configure exception handling in batch and stream processing

Create a Fail activity with UI

1. Search for and drag a Fail activity to the pipeline canvas.
2. Edit its details to enter a failure message and error code.

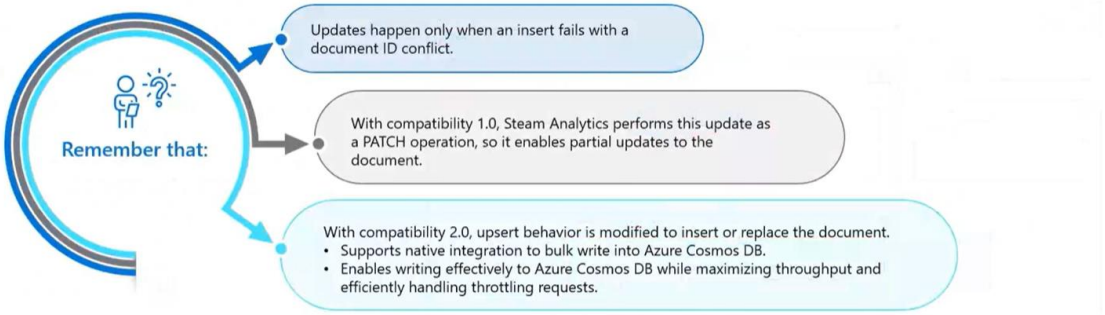
Understand the Fail activity error code

Error messages and codes are set by the developer or ADF.



Upsert data in stream processing

With the integration of Stream Analytics with Azure Cosmos DB, you can insert or update records in your container based on a given **Document ID** column.



Remember that Azure Stream Analytics (ASA) only supports appending rows to SQL outputs. You can use workarounds to enable UPDATE, UPSERT, or MERGE on SQL databases, with Azure Functions as the intermediary layer.

Replay archived stream data

Azure Stream Analytics allows checkpoint and replay features, which may have an impact on job recovery. The **replay catch-up time** estimates the length of the delay due to a service upgrade.

To estimate the replay catch-up time:

1. Load the input

Load the input Event Hubs with enough data to cover the largest window size in your query, at expected event rate.

2. Start the job

Start the job using **Now** as the start time.

3. Measure time

Measure the time between the start time and the time when the first output is generated.

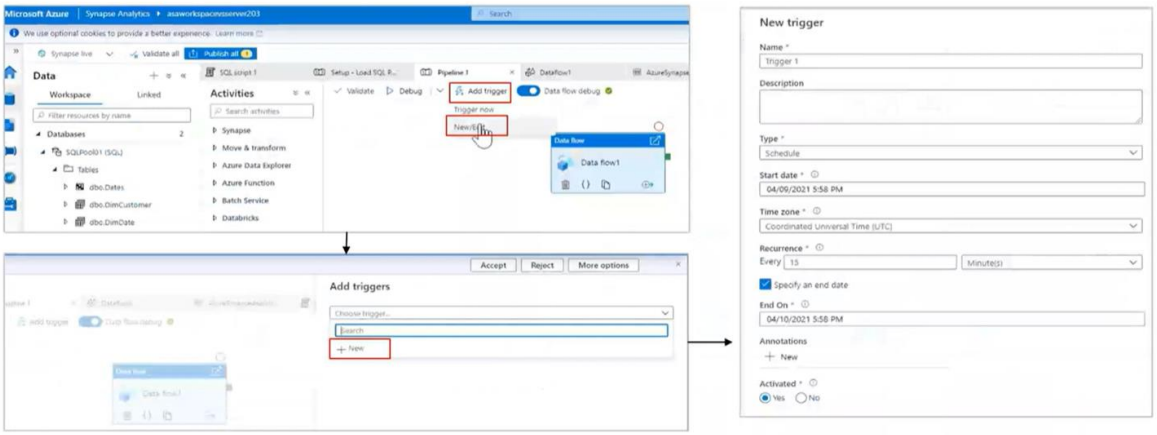
4. Partition job, if necessary

Partition your job if the time is too long. You can also reduce window size.

For events beyond the retention period, we can use Event Hubs Capture to store data in the Azure Blob Store or Data Lake Store.

Trigger batches

You can add a trigger by adding the relevant details in the **New trigger** dialog box.



Handle failed batch loads

Pool and node failures can occur in the background operations and need to be detected and avoided.

Pool errors can be due to resize timeout or failure, automatic scaling failure, or pool deletion failure. Some issues and their fixes include:

- **Resize timeout or failure**
  - Resize timeout too short: Resize timeout.
  - Insufficient core quota: Increase core quota to allocate more nodes.
  - Insufficient subnet IPs when a pool is in a virtual network: Ensure each node has an IP address
  - Insufficient resources when a pool is in a virtual network: Ensure subscription quotas are sufficient for all resources.
  - Large pools with custom VM images: Follow the limitations.
- **Automatic scaling failures**: Use the **autoScaleRun** property to get information on any errors.
- **Pool deletion failures**: Take action depending on the issue.

Node errors include any errors which render the nodes unhealthy and unable to run. Some issues include:

- Start task failures
- Application package download failure
- Container download failure
- Node OS updates
- Node in unusable state
- Node agent log files
- Node disk full

Since these errors can occur even when the batch successfully allocates nodes in a pool and render nodes unusable, it is important to:

- Know the status of nodes being used (and which are unusable) to avoid incurring wasted cost.
- Have a back-up for relevant objects, such as containers, application packages, and start tasks.

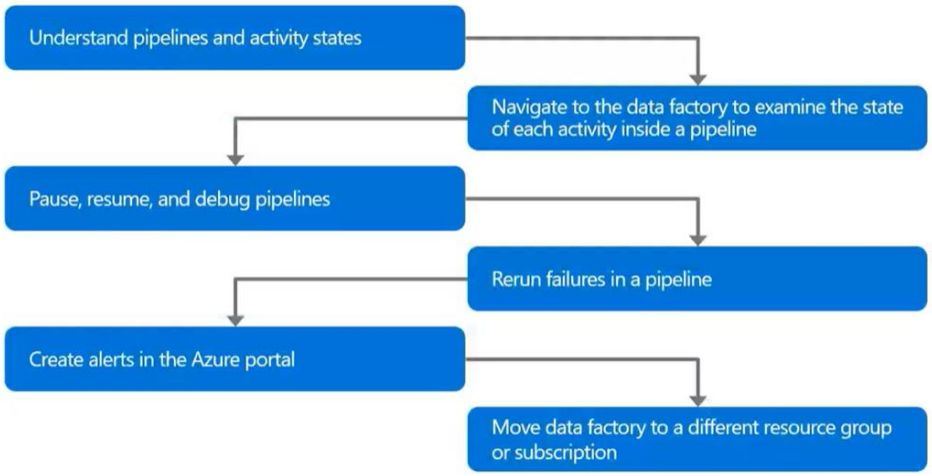
Validate batch loads

There are three ways to validate batch loads:

Subscription-level validation	Operational event data is collected in several categories by the Azure activity log at the subscription level.
Batch account-level validation	For example, each bank account is validated using the Azure Monitor feature.
Batch-resource validation	Batch APIs are used to monitor or query the status of resources in batch applications.

Manage data pipelines in ADF or Azure Synapse Pipelines

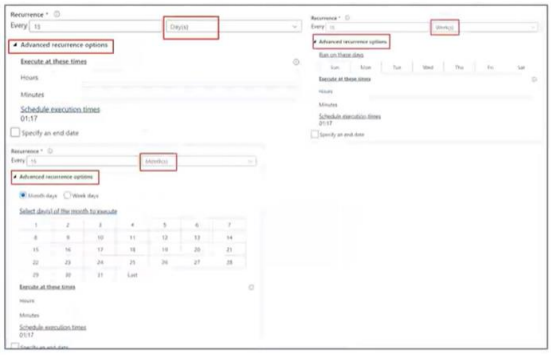
The high-level steps to monitor data pipelines are as follows:



Schedule data pipelines in ADF or Azure Synapse Pipelines

Perform the following steps to schedule data pipelines in ADF or Azure Synapse Pipelines:

1. Access the **Edit** tab in ADF or Azure Synapse Pipelines to select the **New/Edit** trigger.
2. From the **Add Triggers** page, select **Choose Trigger** and **+New**.
3. From the **New Triggers** page, specify the details of the schedule.
4. Publish the trigger using **Publish All** to start triggering and pipeline runs.
5. Switch to the **Pipeline runs** tab refresh the list for pipeline runs.



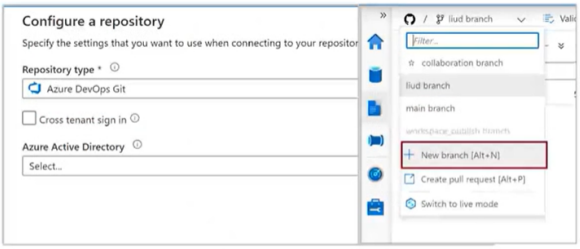
You can also schedule data pipelines through Powershell, CLI, .NET SDK, and REST API.

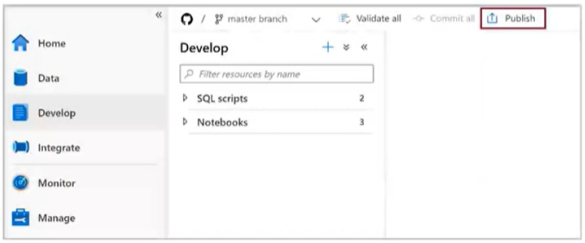
Implement version control for pipeline artifacts

With Synapse Studio, you can associate the synapse workspace with a Git repository, Azure DevOps, or GitHub.

To implement version control, you need to:

- **Create feature branches**: Use the default collaboration branch or create feature branches in Synapse Studio (click + **New Branch** in the branch dropdown).
- **Configure publishing settings**: Use the workspace templates in the workspace_publish branch (generated by Synapse Studio) or configure a custom publish branch (add a publish_config.json file to the root folder in the collaboration branch).
- **Publish code changes**: Use the **Publish** option to manually publish code changes in the collaboration branch (after merging changes to it) to the Synapse service.

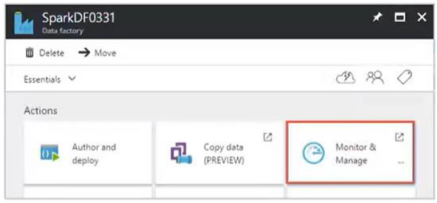


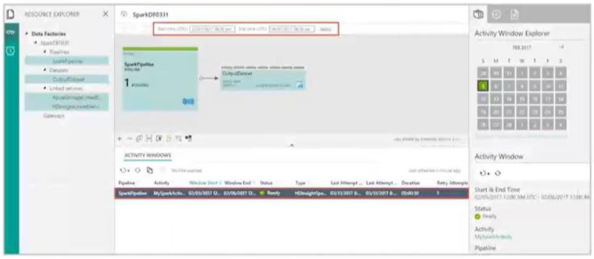


Manage Spark jobs in a pipeline

The steps to manage Spark jobs in a pipeline are:

- From the **Data Factory** page, select **Monitor & Manage**. This starts the monitoring application in another tab.
- Reset the **Start time** filter and click **Apply**.
- Ensure that the data slice is in the **Ready** state.
- From the **ACTIVITY WINDOWS** section, select an activity to explore its details.





Question 01

You need to use Spark to analyze data in a parquet file. What should you do?

Choose the correct answer

1. Load the parquet file into a dataframe
2. Import the data into a table in a serverless SQL pool
3. Convert the data to the CSV format

Question 02

You have loaded a Spark dataframe with data, and you now want to use it in a Delta Lake table. What format should you use to write the dataframe to storage?

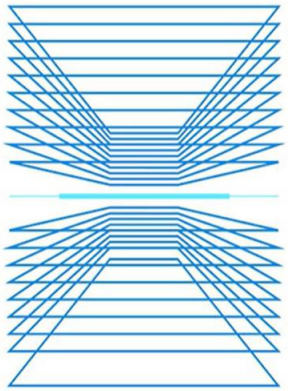
Choose the correct answer

1. CSV
2. PARQUET
3. DELTA

Review (1/3)

Recapping what we covered in Functional Group 2

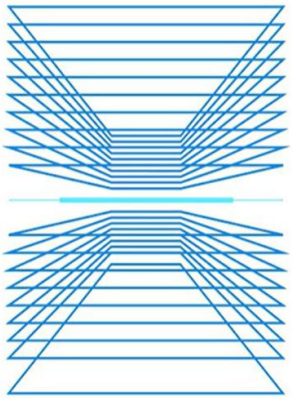
- Transform data by using Transact-SQL (T-SQL)
- Ingest and transform data by using Azure Synapse Pipelines or ADF
- Design and implement incremental loads
- Transform data by using Apache Spark
- Transform data by using Azure Stream Analytics
- Cleanse data
- Handle duplicate, missing, and late-arriving data
- Split data
- Shred JSON
- Encode and decode data
- Configure error handling for a transformation
- Normalize and denormalize values
- Perform data exploratory analysis



Review (2/3)

Recapping what we covered in Functional Group 2

- Develop batch processing solutions by using Azure Data Lake Storage, Azure Databricks, Azure Synapse Analytics, and ADF
- Use PolyBase to load data to a SQL pool and configure the batch size
- Implement Azure Synapse Link and query the replicated data
- Create data pipelines
- Integrate Jupyter or Python notebooks into a data pipeline
- Upsert data in batch processing
- Configure batch retention and revert data to a previous state
- Read from and write to a delta lake



Review (3/3)

Recapping what we covered in Functional Group 2

- Create a stream processing solution
- Process data by using Spark structured streaming and create windowed aggregates
- Handle schema drift and process time series data
- Process data across partitions and within one partition
- Configure watermarking during processing
- Scale resources and create tests for data pipelines in stream processing
- Optimize pipelines for analytical or transactional purposes
- Handle interruptions and configure exception handling in stream processing
- Upsert data in stream processing
- Replay archived stream data, trigger batches, and handle failed batch loads and validate batch loads
- Manage and schedule data pipelines in ADF or Azure Synapse Pipelines
- Implement version control for pipeline artifacts
- Manage Spark jobs in a pipeline