**Advanced ADF Data Flow Concepts – Study Guide**

**1. Flowlet Reuse**

**Overview:**

Flowlets in Azure Data Factory are reusable components within Data Flows. They allow you to encapsulate a group of transformations and reuse the logic across multiple data flows.

**Key Benefits:**

* **Modular Development**: Reduces duplication of transformation logic.
* **Reusability**: Can be reused in multiple Data Flows.
* **Maintainability**: Centralized updates in one place.

**When to Use:**

* Repeated transformations like standardization, data cleaning, or field calculations across datasets.

**How to Implement:**

1. Author → **+ New Flowlet**.
2. Add **Input**, transformation logic (e.g., Derived Column), and **Output**.
3. In any Data Flow, use **Add Flowlet** to include it.

**Example Use Case:**

Create a SalaryNormalizer flowlet that replaces non-numeric salaries with -1.

**2. Derived Column & Conditional Split**

**Derived Column**

**Purpose:**

Used to create new columns or modify existing columns using expressions.

**Common Scenarios:**

* Data standardization (upper/lower case, trimming).
* Calculations (e.g., tax, normalized score).
* Conversions (date formats, null handling).

**Example Expression:**

NormalizedSalary = iif(isNull(toInteger(Salary)), -1, toInteger(Salary))

**Conditional Split**

**Purpose:**

Allows routing of rows to different outputs based on conditions.

**Use Cases:**

* Separating valid and invalid data.
* Routing rows by category (e.g., department, region).
* Building different transformation pipelines based on type.

**Example Conditions:**

* Department == 'IT'
* Salary > 80000

**3. Assert-Row (Data Quality) Transform**

**Overview:**

The Assert transformation helps enforce **data quality rules**. You can define conditions that must be met, and rows that fail are dropped or flagged.

**Types of Assertions:**

* **Assert**: Fails the pipeline if assertion fails.
* **Warning**: Logs a warning but continues processing.
* **None**: No impact, used for monitoring.

**Use Cases:**

* Rejecting rows with missing critical data (e.g., ID is null).
* Validating value ranges (Salary > 0).
* Verifying data types.

**Example:**

Assert NormalizedSalary > 0

**4. Sink Staging & Partitioning**

**Staging in Sink**

**Purpose:**

Improves performance and reliability by writing data to a temporary location (staging) before final write.

**When to Enable:**

* Large data volume writes.
* Sink is a slow endpoint like Azure SQL.
* Need retry and buffering mechanism.

**Partitioning**

**Purpose:**

Improves **write performance** by splitting data into manageable chunks.

**Types:**

* **Round Robin**
* **Hash Partition** (by a specific column)
* **Key Partition** (by value like year, region)

**Use Case:**

Partition Sink by JoinYear to separate files by year.

**5. Debug & Data Preview**

**Debug Mode:**

**Features:**

* Enables real-time data preview during design.
* Uses a Spark cluster behind the scenes.
* Helps spot data errors early.

**Usage Tips:**

* Enable “Debug” before previewing.
* Keep sample size small to reduce cost.
* Use **Assertions + Data Preview** for quality checks.

**Data Preview:**

Lets you inspect:

* Schema
* Row values
* Transformation outputs
* Row counts

**6. Performance Tuning**

**Key Techniques:**

1. **Partitioning**: Optimize large data writes and transformations.
2. **Caching**: Use cache transform where reusing expensive lookups.
3. **Reduce Shuffling**:
   * Avoid Derived Columns before Filter/Split if not needed.
   * Partition before Join/Union to reduce overhead.
4. **Use Selective Columns**: Avoid loading unnecessary columns.
5. **Disable Data Flow Debug After Design**: Saves cost.
6. **Source Pushdown**: Use **Query folding** wherever possible (e.g., in SQL sources).

**Monitoring Tools:**

* ADF Monitor → Pipeline Run → Data Flow Details.
* Activity duration, shuffle size, memory usage, and skew indicators.

**Suggested Best Practices**

| **Topic** | **Best Practice** |
| --- | --- |
| Flowlets | Use for repeated logic (standardize columns, parse fields). |
| Derived Column | Do early in flow to avoid recomputation. |
| Conditional Split | Put early to split bad/good data paths. |
| Assert-Row | Set as "warning" in dev, "assert" in prod. |
| Sink Partitioning | Partition by date, region, or size-related fields. |
| Debug | Use sparingly; always disable after preview. |
| Performance Tuning | Monitor shuffle, partition skew, and broadcast joins using ADF monitoring UI. |

**Wrap-Up**

Mastering these advanced Data Flow transformations helps you build **scalable**, **modular**, and **high-performance** pipelines in Azure Data Factory. Always aim to:

* Modularize with Flowlets
* Validate early with Assert-Row
* Filter and split smartly
* Partition sinks for performance
* Tune with debugging insights