Top 50 ADF Real-Time Scenarios

(Full Explanation + How-To + Example)

Aditya Raja

https://www.linkedin.com/in/adityaraja853/

Top 50 ADF Real-Time Scenarios
(Full Explanation + How-To + Example)

🔹 Scenario 1 — Load Only New/Modified Records (Incremental Load / Delta Load)

📌 Explanation

Most pipelines should not reload full tables. Use a **watermark** column (`LastUpdated`, `ModifiedDate`) to load only changed rows.

📌 How To Implement

1. Create Watermark table.
2. Lookup last watermark value in ADF.
3. Use dynamic SQL in Copy Activity:
4. `SELECT * FROM Sales`
5. `WHERE ModifiedDate > @{activity('GetWatermark').output.firstRow.lastValue}`
6. Run Copy → update watermark using Stored Proc.

📌 Real Example

Last watermark = **2025-01-10 12:00**
Pipeline loads only rows newer than that value.

🔹 Scenario 2 — Trigger Pipeline Only When File Exists

📌 Explanation

Avoid failures when file hasn't arrived yet.

📌 How To Implement

1. Use **Get Metadata** → check `ChildItems`.
2. Use **If Condition**:
   - TRUE → Copy Activity
   - FALSE → Log + Skip / Send alert

📌 Real Example

If `orders_20250114.csv` exists → process
Else → send Teams/Email alert.

🔷 Scenario 3 — Trigger Pipeline When File Arrives (Event-Based Trigger)

📌 Explanation

ADF supports automatic trigger using **BlobCreated** events.

📌 How To Implement

1. Create Event Trigger.
2. Select Storage Account + Container.
3. Add filter path `/incoming/orders`.

📌 Example

When a new file arrives in `adls/raw/orders/`, pipeline runs instantly.

🔷 Scenario 4 — Pass Parameters from ADF to Databricks Notebook

📌 Explanation

Dynamic pipelines need to instruct Databricks which file/date/table to process.

📌 How To Implement

ADF Notebook Activity:

```
baseParameters:
  fileDate: @pipeline().parameters.fileDate
  objectName: @pipeline().parameters.tableName
```

Inside notebook:

```
fileDate = dbutils.widgets.get("fileDate")
table = dbutils.widgets.get("objectName")
```

📌 Example

fileDate = `2025-01-14`
Databricks loads only that partition from ADLS.

🔷 Scenario 5 — Implement SCD Type-2 (Maintain Data History)

📌 Explanation

SCD2 maintains full history when dimension data changes.

📌 How To Implement

In **Mapping Data Flow**:

- Use **Alter Row**:
  - INSERT when new
  - UPDATE when data changed
- Add:
  - EffectiveStartDate
  - EffectiveEndDate
  - IsCurrent flag

📌 Example

Customer moved from "Delhi → Bangalore"
Old record: EndDate updated
New record inserted with IsCurrent = 1.

🔷 Scenario 6 — Merge Incremental Changes Using Databricks (Delta MERGE)

📌 Explanation

Databricks Delta supports ACID merges for incremental updates.

📌 How To Implement

```
MERGE INTO silver.customers AS tgt
USING bronze.customers AS src
ON tgt.id = src.id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
```

📌 Example

Used when daily updates need to sync with Silver layer.

🔷 Scenario 7 — Handle Schema Drift Automatically

📌 Explanation

New columns appear frequently in JSON/CSV files.

📌 How To Implement

In **Mapping Data Flow**:

- Enable Schema Drift
- Use Auto-Map
- Derive missing columns if needed

📌 Example

Incoming JSON adds new field `device_type` → pipeline continues without failing.

🔷 Scenario 8 — Load Nested JSON Files

📌 Explanation

ADF Mapping Dataflow supports flattening nested JSON.

📌 How To Implement

1. Use **Flatten Transformation**.
2. Select nested arrays/objects.
3. Map flattened fields to sink.

📌 Example

JSON:

```
customer.address.city
```

Flatten → Create column: `city`.

🔷 Scenario 9 — Optimize Copy Activity for Large Data (10M+ Records)

📌 Explanation

To speed up SQL → ADLS or ADLS → Synapse loads.

📌 How To Implement

- Enable **parallel copy**
- Increase **DIUs**
- Use **partition option**
- Use **staging** for SQL → Synapse

📌 Example

Copy 20M rows from SQL → ADLS in 12 minutes using 16 parallel copies.

🔷 Scenario 10 — Process Multiple Files in Folder Dynamically

📌 Explanation

ADF must ingest multiple files automatically.

📌 How To Implement

1. Get Metadata → ChildItems
2. ForEach → loop through file list
3. Use dynamic file path in Copy:
4. `@item().name`

📌 Example

Folder contains:

- sales_20250101.csv
- sales_20250102.csv
- sales_20250103.csv

Pipeline loads all files one by one.

🔷 Scenario 11 — Process millions of rows efficiently (SQL → ADLS / Synapse)

📌 Scenario

You need to move **tens of millions of rows** daily from SQL to ADLS/Synapse and the pipeline is very slow.

📌 Explanation

For large volumes, **parallelism + partitioning + proper sink** (Parquet/PolyBase) are key. A single-threaded copy or row-by-row insert will be too slow.

📌 How to handle

1. **Use PolyBase / Bulk insert** when loading into Synapse.
2. **Partition the data** (date, id ranges) and copy per partition.
3. Use **Copy Activity** with:
   - `maxConcurrentConnections`
   - `parallelCopies`
4. Prefer **Parquet** or **Delta** instead of CSV.
5. Use **staging** (e.g., Blob → Synapse via PolyBase).

📌 Example

- In Copy Activity → **Source query**:

```
SELECT * FROM Sales
WHERE OrderDate >= @startDate
  AND OrderDate < @endDate;
```

- In pipeline:
  - Use a **ForEach** over date partitions (e.g., last 7 days).
  - Each iteration calls the Copy Activity with different `startDate/endDate`.

🔷 Scenario 12 — Dynamic file-name based ingestion (daily files)

📌 Scenario

Every day, a new file like `sales_2025-11-23.csv` lands in ADLS. You must build **one reusable pipeline** that picks today's file.

📌 Explanation

You can use **pipeline parameters + dynamic content** to avoid hardcoding file names.

📌 How to handle

1. Create a **pipeline parameter**: `fileDate`.
2. Default value: `@formatDateTime(utcNow(), 'yyyy-MM-dd')`

3. In your **dataset file path**, use:

```
@concat('sales_', pipeline().parameters.fileDate, '.csv')
```

4. Trigger the pipeline **daily** with a schedule trigger.

📌 Example

- Dataset → File name: `@concat('sales_', pipeline().parameters.fileDate, '.csv')`
- If you need to re-run for a specific day, manually trigger and **override `fileDate`** (e.g. `2025-11-20`).

🔷 Scenario 13 — Call REST API with pagination (multiple pages of data)

📌 Scenario

A REST API returns only **1000 records per page** and uses a `nextPageToken` for pagination. You must **load all pages** into ADLS or SQL.

📌 Explanation

You use **Web Activity + Until loop** to repeatedly call the API until there's no next page.

📌 How to handle

1. Start with an initial Web Activity to call Page 1.
2. Store `nextPageToken` in a **pipeline variable**.
3. Use an **Until** activity:
   - Condition: `@equals(variables('nextPageToken'), null)`
4. Inside **Until**:
   - Web Activity calls API with `nextPageToken`.
   - Append response to ADLS via Copy Activity (REST → ADLS).
   - Update `nextPageToken` variable from response.

📌 Example

- URL in Web Activity:

```
@concat('https://api.example.com/users?pageToken=', variables('nextPageToken'))
```

- Variable update:

```
setVariable:
Name: nextPageToken
Value: @activity('CallAPI').output.nextPageToken
```

🔷 Scenario 14 — Process all files in a folder (loop through multiple files)

📌 Scenario

A folder `/raw/sales/` contains **multiple daily files**. You must process **all files automatically**.

📌 Explanation

Use **Get Metadata** to list files and then **ForEach** to loop through each file name.

📌 How to handle

1. **Get Metadata** activity:
   - Point to folder `/raw/sales/`
   - Set **Field list** = `Child items`
2. **ForEach** activity:
   - Items: `@activity('Get Metadata1').output.childItems`
3. Inside ForEach:
   - Copy Activity with dataset file path:
     - Directory: `/raw/sales/`
     - File name: `@item().name`

📌 Example

Dataset file field:

```
@item().name
```

This will copy/process each file like `sales_2025-11-20.csv`, `sales_2025-11-21.csv`, etc, without hardcoding.

🔷 Scenario 15 — Prevent parallel pipeline runs (no overlap allowed)

📌 Scenario

Your daily pipeline should **not run if the previous run is still running** (to avoid double-processing).

📌 Explanation

Use **pipeline concurrency** setting so that ADF **queues** new runs instead of running them in parallel.

📌 How to handle

1. Go to **pipeline Settings**.
2. Set **Concurrency** = `1`.
3. If a new trigger fires while pipeline is running, the new run is **queued**, not executed in parallel.

📌 Example

- Daily trigger at 1 AM.
- Suppose 1 AM run takes 2 hours.
- If you manually trigger the same pipeline at 1:30 AM, the second run will wait until the first finishes.

🔷 Scenario 16 — Audit logging for every run (who, when, status)

📌 Scenario

You need an **audit log** table capturing: pipeline name, runId, startTime, endTime, status, errorMessage.

📌 Explanation

Use a **Stored Procedure activity** or **Copy to log table** at the end (or on failure) of the pipeline.

📌 How to handle

1. Create a SQL table `PipelineAuditLog`.
2. At the end of the pipeline, use **Stored Procedure** activity:
   - Pass:
     - `@pipeline().RunId`
     - `@pipeline().Pipeline` (name)
     - `@utcNow()` for end time
     - `activity('YourActivity').Status` or full output.
3. For failures:
   - Use an extra activity in **On Failure** path to log error details.

📌 Example

Stored procedure:

```
CREATE PROCEDURE usp_InsertPipelineLog
(
  @RunId NVARCHAR(100),
  @PipelineName NVARCHAR(200),
  @Status NVARCHAR(50),
  @StartTime DATETIME2,
  @EndTime DATETIME2,
  @ErrorMessage NVARCHAR(MAX)
)
AS
BEGIN
  INSERT INTO PipelineAuditLog
  VALUES (@RunId, @PipelineName, @Status, @StartTime, @EndTime, @ErrorMessage);
END
```

🔷 Scenario 17 — Validate file format / schema before processing

📌 Scenario

If the **file has wrong number of columns** or **wrong extension**, the pipeline should **fail early** and not load junk.

📌 Explanation

Use **Get Metadata + If Condition** and optionally **Data Flow Assert** / schema validation.

📌 How to handle

1. **Get Metadata** on file:
   - Fields: `columnCount` (for some sources), `itemName`, `size`.
2. **If Condition**:
   - Example condition: `@equals(endsWith(activity('Get Metadata').output.itemName, '.csv'), true)`
3. Inside True (valid): run Copy/Dataflow.
4. Inside False: call **Fail Activity** or log error.

📌 Example

- Expression to check extension:

```
@equals(last(split(activity('Get Metadata1').output.itemName, '.')), 'csv')
```

If not `.csv`, pipeline goes to error handling path.

🔷 Scenario 18 — Process large JSON files via Databricks (or Mapping Data Flow)

📌 Scenario

You receive **very large JSON files** with nested structure. Mapping Data Flow struggles or is too expensive; you prefer Databricks.

📌 Explanation

Databricks **Auto Loader** or Spark can handle large JSON with **schema inference + incremental load**.

📌 How to handle (Databricks)

1. ADF **Notebook Activity** triggers Databricks notebook.
2. In notebook:

```
df = spark.read.json("abfss://raw@youraccount.dfs.core.windows.net/json/")
# Flatten, transform, filter
df_flat = df.select("id", "name", "address.city", "address.country")
df_flat.write.format("delta").mode("append").save("/mnt/silver/customers")
```

3. ADF orchestrates this as part of pipeline, before loading to Synapse/Gold.

📌 **Example**

- ADF pipeline:
    - Copy file from landing → raw.
    - Notebook Activity to process and write into **Delta tables**.

🔷 Scenario 19 — Copy from on-prem Oracle/SQL Server to ADLS using Self-Hosted IR

📌 Scenario

Your data source is **on-prem SQL/Oracle**, not accessible over public internet. You must securely copy data to ADLS.

📌 Explanation

You need a **Self-Hosted Integration Runtime (SHIR)** installed inside the network/VPN.

📌 How to handle

1. Install **Self-Hosted IR** on a VM or on-prem server with DB access.
2. In ADF, create **Linked Service (SQL/Oracle)** using SHIR.
3. Use Copy Activity:
    - Source: on-prem SQL/Oracle (linked with SHIR).
    - Sink: ADLS Gen2.
4. Use **SQL query** or **table** selection.

📌 Example

Source dataset uses Linked Service `OnPremSql_LS` with IR type = `SelfHostedIntegrationRuntime1`.

SQL query:

```
SELECT * FROM Orders WHERE OrderDate >= @lastWatermark;
```

🔷 Scenario 20 — Stored procedure driven ETL (using SQL logic inside ADF pipeline)

📌 Scenario

You want the **business/merge logic to stay in SQL** (SP) while ADF handles orchestration.

📌 Explanation

ADF can call **Stored Procedures** as part of the pipeline to execute complex logic like SCD2, temp tables, cleanup, etc.

📌 How to handle

1. Create a **Stored Procedure activity**.
2. Linked Service → your SQL DB.
3. Pass parameters (e.g., `RunDate`, `Watermark`) from pipeline.
4. SP does:
   - staging → target table merge
   - update watermark table
   - log row counts.

📌 Example

- In ADF Stored Procedure activity parameters:

```
RunDate = @utcNow()
Watermark = @pipeline().parameters.watermark
```

- Sample SP pseudocode:

```
CREATE PROCEDURE usp_ETL_Sales (@RunDate DATETIME2, @Watermark DATETIME2)
AS
BEGIN
  -- Load from staging to final table with MERGE
  MERGE dbo.SalesFinal AS T
  USING dbo.SalesStaging AS S
  ON T.Id = S.Id
  WHEN MATCHED THEN UPDATE SET ...
  WHEN NOT MATCHED THEN INSERT (...);

  UPDATE ControlTable SET LastWatermark = @RunDate WHERE ProcessName='Sales';
END
```

🔷 Scenario 21 — Partition data in Mapping Data Flows for performance

📌 Explanation (What & Why)

When you process large datasets in Mapping Dataflows, a single partition can become a bottleneck. Partitioning lets ADF split the data (e.g., by date or ID) so Spark can process partitions in parallel → faster pipelines and better cluster utilization.

📌 How to implement in ADF (Steps)

1. Open your **Mapping Data Flow**.
2. Go to the **Optimize** tab.
3. Under *Partitioning*, choose:
   - **Current partitioning** (default), or
   - **Set partitioning** → choose **Hash**, **Range**, or **Key**.
4. Select a column to partition by (e.g., `OrderDate`, `CustomerId`).

5. Set number of partitions (or let ADF decide based on compute).
6. Publish and test with debug to compare performance (before vs after).

📌 Example you can say in an interview

"We had a big facts table with ~50M rows. The Dataflow was slow because everything was processed in a single partition. I enabled partitioning in the Optimize tab and partitioned by `OrderDate` month. That allowed Spark to process multiple partitions in parallel and reduced Dataflow execution time by around 35–40%."

🔷 Scenario 22 — Copy activity fails due to timeout or long run time

📌 Explanation (What & Why)

When copying huge data (GBs/TBs), Copy Activity can fail with timeouts, throttling, or slow performance. You must tune performance and sometimes use **staged copy** or **PolyBase/Bulk** to offload work.

📌 How to implement in ADF (Steps)

1. Open **Copy Activity** → **Settings**.
2. Increase **Timeout** (e.g., from default to 2–4 hours for heavy loads).
3. In **Performance**:
   - Enable **Parallel copy** (e.g., 16–32).
   - Set **Data integration units (DIUs)** higher if needed.
4. For SQL → Synapse loads, enable **Staging (PolyBase)**:
   - Use **Blob/ADLS** as staging.
   - Check *Use PolyBase* or *Bulk insert* where available.
5. Ensure network bandwidth / SHIR VM size is sufficient.
6. Re-run and monitor throughput (MB/s) from Monitor tab.

📌 Example interview answer

"Our SQL-to-ADLS copy was failing with timeout due to 200M+ rows. I increased the activity timeout, enabled parallel copy with 16 parallel threads, and used staged copy with PolyBase into Synapse. That increased throughput and eliminated timeouts."

🔷 Scenario 23 — Process near real-time/streaming data with ADF

📌 Explanation (What & Why)

ADF is mainly batch-oriented, but you can achieve *near real-time* by combining **Event Hub** or **IoT Hub** with frequent-trigger pipelines or using Dataflows on streaming landing data.

📌 How to implement in ADF (Steps)

1. Use **Event Hub / IoT Hub** to receive streaming messages.
2. Configure **Capture** to write events into **ADLS/Blob** in small files (e.g., every 1–5 minutes).
3. In ADF, create a **Tumbling Window Trigger** (e.g., every 5 minutes).
4. Pipeline steps:
    - Read newly landed captured files from ADLS.
    - Use **Mapping Data Flow** or **Databricks** to clean/transform.
    - Write results to **Silver/Gold** layer or Synapse.
5. Implement **watermark/last processed time** to avoid double-processing.

📌 Example interview answer

"For IoT telemetry, events landed in Event Hub and were captured to ADLS every 5 minutes. I created a tumbling window trigger in ADF that picked up the latest batch, transformed it in Dataflows, and loaded curated data into Synapse. This gave us near real-time dashboards with a 5-minute delay."

🔷 Scenario 24 — Load CSV files with varying number of columns (schema drift)

📌 Explanation (What & Why)

In real projects, CSV files may evolve — new columns added, some removed. This is called **schema drift**. If you rigidly define columns, pipelines will fail on schema changes. ADF Dataflow can handle schema drift dynamically.

📌 How to implement in ADF (Steps)

1. In **Mapping Data Flow**, create a **Source** pointing to your CSV.
2. In source options, enable **Allow schema drift**.
3. Use `Select` or `Derived Column` to map only required columns.
4. Use `Column patterns` if you want to handle "all extra columns" generically.
5. Sink:
    - Use a flexible format like **Parquet** or **Delta** which supports schema evolution.
6. Test with old and new versions of the files.

📌 Example interview answer

"Client CSV files kept changing with extra columns. To avoid frequent failures and rework, I enabled schema drift in Mapping Data Flow and mapped only the required core columns, while still landing all new columns into a 'flexible' parquet sink for exploration."

🔷 Scenario 25 — Append data instead of overwriting existing data

📌 Explanation (What & Why)

Sometimes you ingest **daily incremental data** and want to **append** to existing data instead of overwriting the full dataset. This is common for logs, transaction feeds, etc.

📌 How to implement in ADF (Steps)

1. In **Copy Activity → Sink settings**:
   - Choose **Copy behavior**: `Merge`, `Upsert`, or `Insert/Append` (options vary by connector).
2. For data lake formats like Parquet:
   - Use file naming strategy with date/time partition (so every run writes new files).
3. For SQL/Synapse:
   - Use **pre-copy script** to filter duplicates if needed.
4. Optionally use **Dataflows or MERGE** if you need upsert logic.

📌 Example interview answer

"For daily transaction loads, we just needed to append new data. I configured the Copy Activity sink to append to existing parquet partitions by date, using dynamic paths like `year=/month=/day=`. This allowed us to keep history without overwriting older data."

🔷 Scenario 26 — Automatically delete old files from data lake (retention policy)

📌 Explanation (What & Why)

Storage costs can explode if you keep all raw data forever. Often you need a **retention policy** (e.g., keep only 90 days of raw files).

📌 How to implement in ADF (Steps)

1. Create a pipeline with these activities:
   - **Get Metadata** (list files in a folder).
   - **ForEach** over the list of files.
   - Inside ForEach → **If Condition**:
     - Compare each file's `lastModified` to `utcNow()` minus X days (e.g., 90).
   - If older → **Delete Activity** to delete that file.
2. Schedule this pipeline with a **daily trigger**.
3. Log deleted files (optional) into a table or log file.

📌 Example interview answer

"To control storage, I built a cleanup pipeline that runs daily. It lists files from the raw folder and deletes anything older than 90 days using Delete Activity. This was fully dynamic and based on file `lastModified` metadata, helping us meet cost and compliance requirements."

🔹 Scenario 27 — Handle NULL / missing values in transformations

📌 Explanation (What & Why)

Nulls can break business logic and cause incorrect aggregates. You often need to **impute** or replace nulls to maintain data quality.

📌 How to implement in ADF (Steps)

1. In **Mapping Data Flow**, add a **Derived Column** transformation.
2. Use expressions like:
   - `iif(isNull(City), 'Unknown', City)`
   - `iif(isNull(SalesAmount), 0, SalesAmount)`
3. For numeric columns, you can default to 0 or average; for string columns, use defined placeholders.
4. Optionally add **Assert** transformation to enforce "non-null" constraints for critical columns.

📌 Example interview answer

"Customer master had many null `City` values, which was affecting segmentation. In Dataflows, I added a Derived Column to replace null cities with 'Unknown' and null numeric fields with 0. This ensured cleaner aggregates and prevented issues in downstream reports."

🔹 Scenario 28 — Combine thousands of small files into fewer large files

📌 Explanation (What & Why)

Too many small files ("small file problem") makes queries slow and costly (especially with Spark/Synapse). Best practice is to **compact** small files into larger partitions.

📌 How to implement in ADF (Steps)

1. Use **Mapping Data Flow** or **Databricks** to read all small files from a folder.
2. In **Sink** settings:
   - Set **File name option** = Single file per partition or pattern.
   - Configure **Partitioning** so you get fewer, larger files.
3. Write data as **Parquet/Delta** with partitioning by date/other columns.
4. Optionally remove or archive the original small files after successful compaction.

📌 Example interview answer

"We were getting thousands of 1–5 KB files per hour from IoT sensors. Queries were very slow, so I created a daily compaction pipeline that read all small files and wrote them into fewer large parquet files partitioned by date. This reduced the file count drastically and improved query performance."

🔷 Scenario 29 — Process weekly data using Tumbling Window trigger

📌 Explanation (What & Why)

Some business jobs run on **weekly cycles**, like weekly sales reports, aggregates, or snapshots. Tumbling Window triggers are perfect for such fixed, non-overlapping intervals.

📌 How to implement in ADF (Steps)

1. Create a **Tumbling Window Trigger**:
   - Set **recurrence** to `7` days.
   - Set **start time** (e.g., every Monday 01:00 AM).
2. In the pipeline, use trigger context functions such as:
   - `windowStart()` and `windowEnd()` for filtering data.
3. In Copy/Dataflow source query:
   - Filter data between `windowStart` and `windowEnd`.
4. Run and verify that each window processes only that specific week's data.

📌 Example interview answer

"We built weekly sales aggregates that had to run every Monday. I used a tumbling window trigger with a 7-day window. Using `windowStart` and `windowEnd`, I filtered data in the source query so each run processed exactly one week of sales, with no overlaps."

🔷 Scenario 30 — Fail pipeline if data volume is below an expected threshold

📌 Explanation (What & Why)

If a feed suddenly drops from millions of rows to a few records, that usually indicates a problem. You can create **data reasonability checks** and fail the pipeline when volume is suspiciously low.

📌 How to implement in ADF (Steps)

1. After a **Copy Activity**, use a **Lookup** / **Stored Procedure** or **Dataflow** to get **row count** of the loaded data.
2. Compare row count with a configured threshold (e.g., from pipeline parameter or config table).

3. Use **If Condition Activity**:
   - If `RowCount < Threshold` → use **Fail Activity** with custom error message.
   - Else → continue pipeline.
4. Optionally send notification (Logic App / Email) when failure occurs.

📌 Example interview answer

"I implemented a data-quality check where, after loading data, we compared row count with a historical threshold. If the count dropped by more than 80%, an If Condition triggered a Fail activity and sent an alert. This prevented bad/partial loads from silently flowing into production."

🔷 Scenario 31 — Reprocess data for last 'N' days (handle late-arriving data)

📌 Explanation

Sometimes data arrives late due to upstream delays (e.g., at night). You need to **reprocess the last 3–7 days** to ensure completeness.

📌 How-To Steps

1. Create a pipeline parameter: `reprocessDays`.
2. Use it in ADF to compute start date:
3. `@addDays(utcNow(), -pipeline().parameters.reprocessDays)`
4. In your SQL query or Dataflow filter:
5. `SELECT * FROM Sales`
6. `WHERE UpdatedOn >= @startDate`
7. Write output to Silver/Gold layers annually.
8. Optionally delete duplicate records using MERGE.

📌 Example for interviewer

"For late-arriving orders, I built a parameterized pipeline that reprocesses the last 3 days using dynamic date filtering. This ensures no missing transactions in downstream analytics."

🔷 Scenario 32 — Handle missing or corrupt files gracefully

📌 Explanation

If a file is corrupt or has zero bytes, ingestion must **fail safely** or **skip** it without breaking the pipeline.

📌 How-To Steps

1. Use **Get Metadata** to read:
    - size
    - columnCount
    - lastModified
2. Add **If Condition**:
    - If size > 0 → process
    - Else → move file to `/error/` folder
3. Use **Copy Activity** to move corrupted files.
4. Log file details (name, timestamp, error reason).

📌 Example for interviewer

"One day, a zero-byte file caused Copy Activity failures. I added a pre-check using Get Metadata → If size=0 → Move to error folder. This made the pipeline fault-tolerant."

🔷 Scenario 33 — Dynamically generate SQL queries using ADF parameters

📌 Explanation

Some pipelines need dynamic WHERE clauses, table names, or partitions based on input.

📌 How-To Steps

1. Create pipeline parameters:
    - `tableName`
    - `startDate`
    - `endDate`
2. Use dynamic SQL in Copy Activity:
3. `@concat(`
4. `  'SELECT * FROM ', pipeline().parameters.tableName,`
5. `  ' WHERE UpdatedOn >= ''', pipeline().parameters.startDate, '''',`
6. `  ' AND UpdatedOn < ''', pipeline().parameters.endDate, ''''`
7. `)`
8. Pass this as a source query.

📌 Example

"We used dynamic content in Copy Activity to extract data for specific tables and date ranges, making the pipeline metadata-driven."

🔷 Scenario 34 — Implement SCD Type 1 (overwrite existing values)

📌 Explanation

SCD1 updates existing records instead of keeping history. Used for non-historical data.

📌 How-To Steps

Using Mapping Dataflow:

1. Source: new data
2. Lookup: existing target data by key
3. Alter Row:
   - When matched → UPDATE
   - When not matched → INSERT

Using Databricks:

```
MERGE INTO dim_customer AS tgt
USING staging_customer AS src
ON tgt.CustomerID = src.CustomerID
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
```

📌 Example

"Customer phone number updates don't need history, so we implemented SCD1 using Delta MERGE in Databricks."

🔷 Scenario 35 — Trigger multiple child pipelines in parallel (fan-out)

📌 Explanation

Useful when ingesting multiple tables or files at once.

📌 How-To Steps

1. Lookup activity fetches list of tables/files.
2. Use **ForEach**:
   - Set **Batch count** (e.g., `10`).
3. Inside ForEach → add **Execute Pipeline** activity.
4. Each child pipeline processes one table/file.

📌 Example

"We ingested 80 tables by using Lookup + ForEach + Execute Pipeline with batch size 10, improving total load time from 6 hours to 1.5 hours."

🔹 Scenario 36 — Build Metadata-driven ADF pipelines

📌 Explanation

You store configurations in a SQL table instead of hardcoding values in ADF. Makes pipelines reusable across many sources.

📌 How-To Steps

1. Create a config table:
2. `SourceTable    TargetPath      LoadType   WatermarkColumn`
3. `-------------------------------------------------------`
4. `Sales          /sales/         Incremental   ModifiedDate`
5. `Customers      /cust/          FullLoad      null`
6. Use Lookup to read config row.
7. Pass values into child pipeline parameters.
8. Apply dynamic transforms:
    - dynamic source table
    - dynamic sink folder
    - dynamic load type

📌 Example

"A single metadata-driven pipeline ingests 30 different tables using config stored in SQL. Zero hardcoding."

🔹 Scenario 37 — Auto-create folders in ADLS based on date

📌 Explanation

Folders like `/year=2025/month=01/day=12/` must be created dynamically for partitions.

📌 How-To Steps

Use dynamic expressions:

```
@concat(
  'year=', formatDateTime(utcNow(), 'yyyy'),
  '/month=', formatDateTime(utcNow(), 'MM'),
  '/day=', formatDateTime(utcNow(), 'dd')
)
```

Set this in Sink dataset → folder path.

📌 Example

“Daily partitions were created dynamically in ADLS using folder path expressions like year/month/day.”

🔷 Scenario 38 — Call an API that requires authentication token

📌 Explanation

Some APIs require an initial token call → then use the token in subsequent requests.

📌 How-To Steps

1. First Web Activity:
   - POST to `/auth/token`
   - Save token to variable:
   - `@activity('AuthCall').output.access_token`
2. Second Web Activity: call actual API:
   - Add header:
   - `Authorization: Bearer @{variables('token')}`
3. Copy Activity loads JSON response to ADLS/SQL.

📌 Example

“To extract data from ServiceNow API, I created a token-generation Web Activity, stored the token, and used it across multiple API calls.”

🔷 Scenario 39 — Load nested JSON into SQL (Flatten + Mapping)

📌 Explanation

SQL cannot store nested JSON, so flattening is required.

📌 How-To Steps

1. Use Mapping Dataflow.
2. Add **Flatten** transformation:
   - Select array/object field.
3. Map flattened columns to SQL table.
4. Use Derived Column to rename nested fields.

📌 Example

Input JSON:

```
{
  "custId": 101,
```

```
  "address": {
     "city": "Delhi",
     "pincode": 110085
  }
}
```

Flatten result:

```
custId | address_city | address_pincode
```

🔷 Scenario 40 — Stop duplicate files from processing twice

📌 Explanation

If the same file lands twice, your logic must **skip** or **overwrite** correctly to prevent duplicate data.

📌 How-To Steps

1. Maintain a table: `ProcessedFileLog`
   - fileName
   - hash
   - processedDate
2. Before processing:
   - Lookup the log table
   - If file exists → skip
   - Else → process
3. After successful run → insert entry in log table.

📌 Example

"I prevented reprocessing by storing both filename and MD5 checksum in a log table. If the same file arrives, the pipeline skips it."

🔷 Scenario 41 — Improve Copy Activity performance (10× faster loads)

📌 Explanation

Copy Activity performance depends on:

- Parallelism
- DIUs
- File format
- Network throughput

- Source query efficiency

📌 How-To Steps

1. Increase **DIUs** (Data Integration Units).
2. Set **parallelCopies** to 8–32 based on source.
3. Use **partitioned SQL queries** for large tables.
4. Choose **Parquet** instead of CSV for sink (10× faster).
5. Enable **staged copy** for SQL → Synapse loads.
6. Use **column pruning** (SELECT only required columns).
7. Use **compression** like snappy/gzip.

📌 Example

"We moved 180M rows from SQL to ADLS. By using Parquet, increasing DIUs, and setting 16 parallel copies, throughput increased from 12 MB/s to 110 MB/s."

🔷 Scenario 42 — Use Data Partitioning for faster processing

📌 Explanation

Partitioning divides data into chunks so Spark processes them in parallel, reducing execution time drastically.

📌 How-To Steps

In **Mapping Dataflow → Optimize tab**:

1. Choose **Set partitioning**.
2. Select **Hash**, **Key**, or **Range** partitioning.
3. Pick a column:
    - `OrderDate`
    - `CustomerId`
4. Optionally define number of partitions.

📌 Example

"Partitioning a 50M row dataset by OrderDate dropped Dataflow runtime from 13 minutes to 5 minutes."

🔷 Scenario 43 — Debug Mapping Dataflows effectively

📌 Explanation

Debugging helps validate transformations and preview data.

📌 How-To Steps

1. Enable **Debug Mode**.
2. Use **Data Preview** at each transformation.
3. Use **Inspect** tab to review schema and data types.
4. Use breakpoints (temporary endpoints).
5. Validate expressions using built-in expression tester.

📌 Example

"I discovered a null handling error in Derived Column by turning on Data Preview and inspecting mismatched values."

🔷 Scenario 44 — Scale out Integration Runtime automatically

📌 Explanation

Azure IR automatically scales based on workload, but SHIR requires manual scaling.

📌 How-To Steps

For Azure IR (Auto-scale):

- No action required; Microsoft manages compute.
- You only adjust **DIUs** during copy.

For Self-Hosted IR:

1. Add extra SHIR nodes.
2. Enable **High Availability** mode.
3. Enable **Auto Update**.
4. Distribute load by mapping multiple pipelines to SHIR nodes.

📌 Example

"To handle heavy on-prem Oracle loads, I added two more SHIR nodes, doubling throughput."

🔷 Scenario 45 — PolyBase vs Bulk Copy for Synapse ingestion

📌 Explanation

Both are used to ingest data into Synapse, but differ in performance and function.

📌 How-To Steps

Feature	PolyBase	Bulk Copy
Performance	Extremely fast for large datasets	Medium-fast
Use case	SQL → Synapse loads	CSV → Synapse loads
Requires staging?	Yes	No
Parallelism	Very high	Limited

In ADF:

- Enable **Use PolyBase** when copying into Synapse from Blob/ADLS.
- Use **Bulk Insert** for smaller but frequent loads.

📌 Example

"PolyBase reduced load time for 40 GB parquet data from 1 hour to 9 minutes."

🔷 Scenario 46 — Use Managed Identity to secure ADF access

📌 Explanation

Managed Identity eliminates passwords and securely authenticates ADF to resources like ADLS, Key Vault, SQL, etc.

📌 How-To Steps

1. Enable **Managed Identity** in ADF.
2. Assign roles to ADF identity:
    - **Storage Blob Contributor**
    - **Key Vault Reader**
    - **SQL DB Contributor**
3. In linked services, choose **Managed Identity authentication**.
4. Remove hardcoded credentials.

📌 Example

"We replaced SQL credentials with Managed Identity, improving security and meeting compliance."

🔷 Scenario 47 — Setup RBAC for Data Factory (Granular permissions)

📌 Explanation

RBAC controls who can edit pipelines, run pipelines, read data, etc.

📌 How-To Steps

Go to **Azure Portal → ADF → Access Control (IAM)**
Assign roles:

- **Owner** — full access
- **Contributor** — create/update pipelines
- **Reader** — view only
- **Data Factory Pipeline Operator** — can run pipelines, not edit

📌 Example

"We restricted junior developers to Pipeline Operator role so they couldn't modify production pipelines."

🔷 Scenario 48 — Centralized monitoring using ADF + Azure Monitor

📌 Explanation

ADF integrates logs & metrics into Azure Monitor for centralized alerting.

📌 How-To Steps

1. Enable **Diagnostic Settings** in ADF:
   - PipelineRuns
   - ActivityRuns
   - TriggerRuns
2. Send logs to:
   - Log Analytics Workspace
   - Event Hub
   - Storage account
3. Build **KQL dashboards** for failures or long-running pipelines.

📌 Example

"We created KQL alerts for pipeline failures. Teams received alerts within 30 seconds of failure."

🔷 Scenario 49 — Implement audit logging for every activity in pipeline

📌 Explanation

Logging ensures traceability and governance.

📌 How-To Steps

1. Use **Activity Output** to extract:
    - runId
    - activityName
    - status
    - start/end time
2. Insert logs into SQL or ADLS via:
    - Stored Procedure
    - Web activity (if hitting API)
    - Copy to lake
3. Use **OnSuccess**, **OnFailure**, and **OnCompletion** paths.

📌 Example

"Each activity wrote log entries to a central audit table. This helped debug failures quickly."

🔷 Scenario 50 — Encrypt data in transit and at rest

📌 Explanation

ADF supports encryption automatically, but you may need to enable specific features for compliance.

📌 How-To Steps

At rest:

- ADLS, Blob, SQL, Synapse → automatically encrypted with AES-256.
- For extra safety, enable **Customer Managed Keys (CMK)**.

In transit:

- ADF enforces **HTTPS/TLS 1.2**.
- Use Private Endpoints to avoid public internet.
- For SQL, enforce TLS connection strings.

📌 Example

"For a finance client, we enabled Customer Managed Keys for ADLS encryption and private endpoints for secure pipeline traffic."