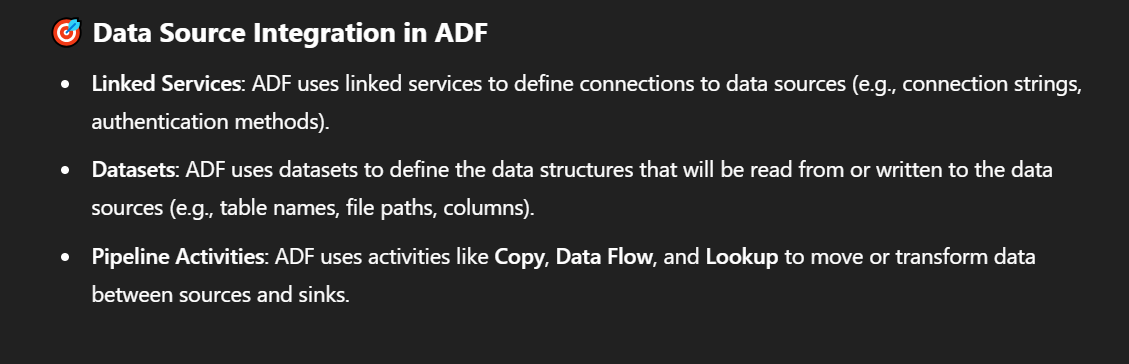
1. What are the main components of Azure Data Factory?

**✅ Main Components of Azure Data Factory**

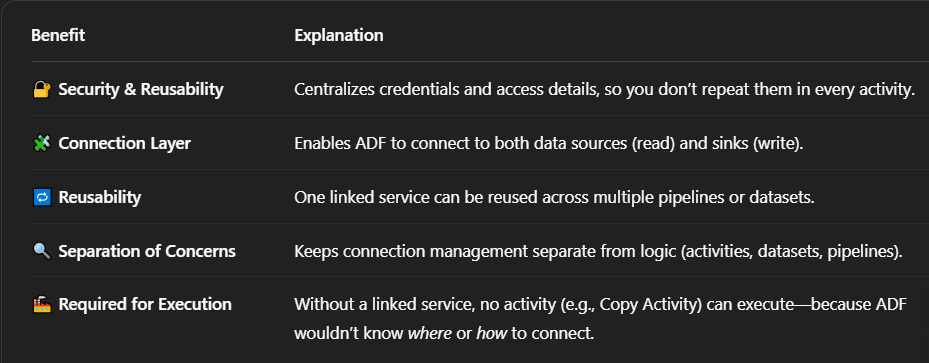
| **Component** | **Description** |
| --- | --- |
| **Pipeline** | A logical group of activities that perform a task. You can think of a pipeline as a workflow or a container. |
| **Activity** | A single step in a pipeline. Activities can move data (Copy Activity), transform data (Data Flow), or control flow (If Condition, ForEach, etc.). |
| **Datasets** | Define the schema and location of the data used by activities. For example, a dataset might point to a table in SQL DB or a file in Blob Storage. |
| **Linked Services** | Like connection strings—define connection info to data sources (e.g., Azure SQL, Blob Storage, REST APIs). |
| **Integration Runtime (IR)** | The compute infrastructure used by ADF to move or transform data. There are three types:  - **Azure IR** (cloud-based)  - **Self-hosted IR** (on-premises or VNet environments)  - **Azure SSIS IR** (for running SSIS packages) |
| **Triggers** | Define when a pipeline should run. Can be scheduled, tumbling window, event-based, or manually triggered. |
| **Data Flows** | Visually-designed, scalable transformation logic for data. Useful for complex data shaping without coding. |
| **Monitoring & Logging** | ADF provides detailed monitoring of pipeline runs, activity failures, and integration runtime metrics. |



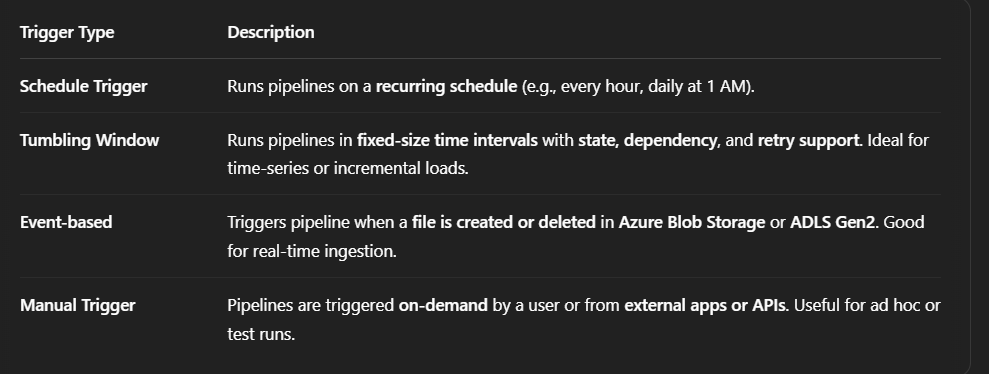
1. What are Linked Services in ADE and why are they important?

A **Linked Service** in ADF is similar to a **connection string** in traditional applications. It tells ADF **how to connect** to external systems like:

* Azure Blob Storage
* Azure SQL Database
* REST APIs
* On-premises SQL Server
* Snowflake, Amazon S3, and more



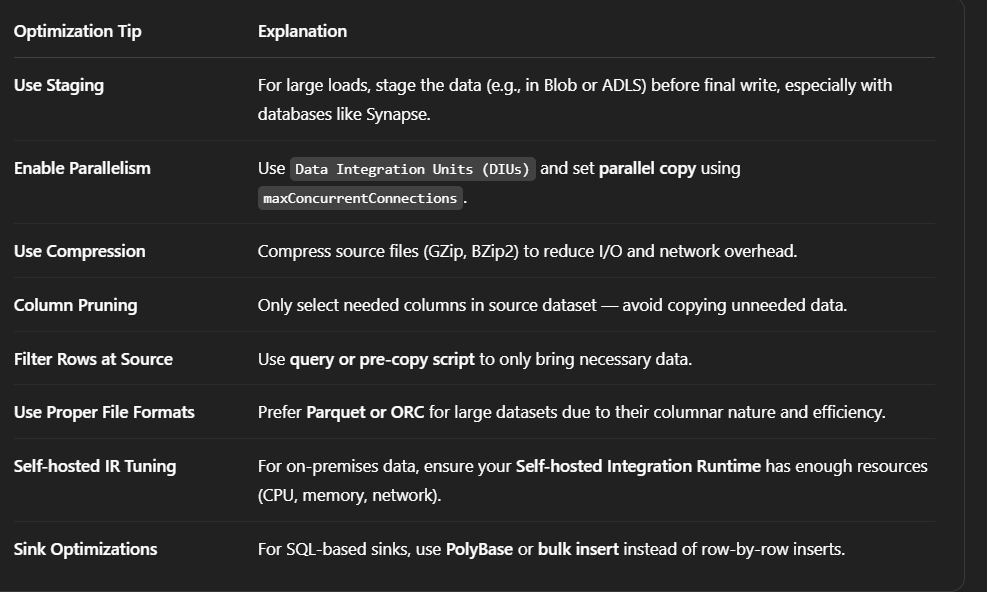
1. What types of triggers are available in ADF for pipeline execution?



1. What is the Copy Activity in ADE and how can you optimize its performance?

The **Copy Activity**:

* Transfers data between various **data stores**
* Supports **structured**, **semi-structured**, and **unstructured** formats
* Works with both **cloud** and **on-premises** sources
* Can transform data **in-flight** (e.g., column mapping, data type conversion)



1. How does ADF handle running multiple pipeline tasks simultaneously ?

**1. Parallel Activities Within a Pipeline**

* You can **run multiple activities in parallel** by not linking them sequentially with dependencies.
* For example, if Activity A doesn't depend on B or C, all three can execute at the same time.

**2. Concurrency Across Pipelines**

* ADF can execute **multiple pipelines at the same time**.
* Each pipeline run is **independent** and doesn’t interfere with others.

**3. ForEach Activity Parallelism**

* The ForEach activity has a **batchCount** property, which controls how many items are processed **in parallel**.

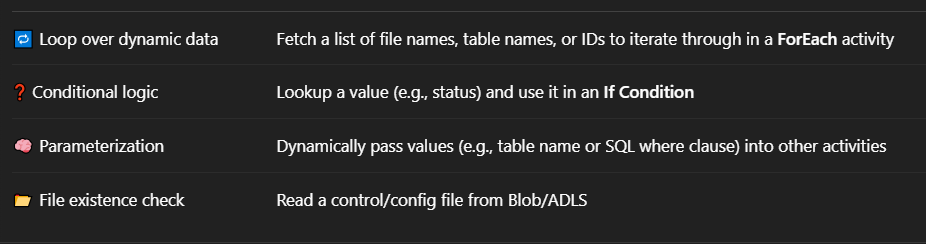
**4. Integration Runtime Scaling**

* ADF uses **Integration Runtime (IR)** to process activities.
  + **Azure IR** scales automatically based on workload.
  + **Self-hosted IR** supports parallelism up to the capacity of the host machine.
  + You can configure **concurrency limits** and **parallel copy** options.

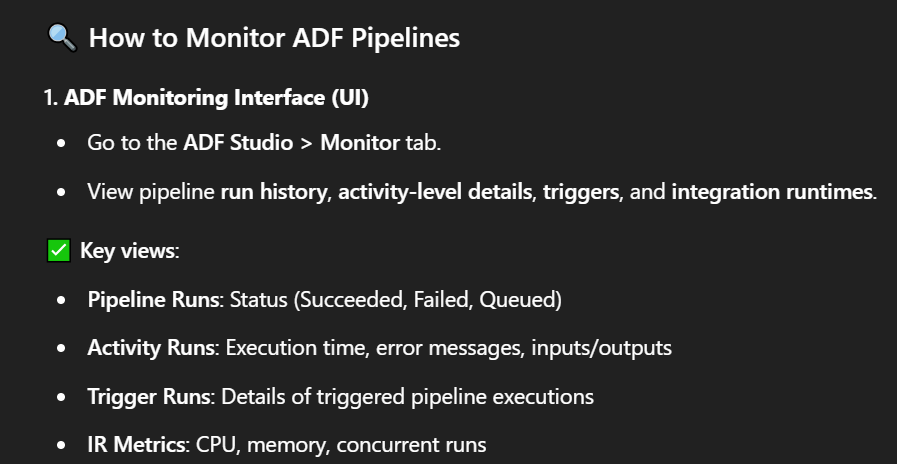
1. What is the Lookup activity in ADE and how is it used?

It executes a **query or reads a file** (CSV, JSON, etc.) and returns:

* A **single row** or
* An **array of rows** (up to 5,000 rows or 4 MB)



1. How do you monitor and troubleshoot ADF pipelines?



1. Tumbling, Sliding and Hopping

**1. Tumbling Window**

* **Fixed-sized**, **non-overlapping**, continuous time windows.
* Every event belongs to **exactly one window**.

**2. Sliding Window**

* **Fixed-size** window that **moves (slides)** at a defined interval.
* Windows **can overlap**.
* Events may appear in **multiple windows**

**3. Hopping Window**

* Like **sliding window**, but with **no strict relation between slide and size**.
* You can define a **window size larger than the hop size**, causing **overlap**.

1. What is the Get Metadata activity in ADF?

The **Get Metadata** activity in **Azure Data Factory (ADF)** is used to **retrieve metadata** from **data sources** like files, folders, databases, or tables. This is particularly useful when building **dynamic pipelines**, where the structure or content of the data source can affect the pipeline's behavior.

**What Metadata Can You Retrieve?**

Depending on the type of data source (e.g., file, folder, table), you can retrieve different metadata fields:

**🗂️ For Files:**

* exists
* size
* lastModified
* itemName

**For Folders:**

* childItems (list of files/folders inside)
* itemType
* exists

**For Tables:**

* schema (list of columns and data types)
* structure
* columnCount

1. How do you manage schema changes in ADF?

**1. Use Schema Drift in Mapping Data Flows**

**Schema drift** allows ADF to handle data with **unknown or changing schema** at runtime.

* Use **Mapping Data Flow**.
* In **source and sink** settings, enable **"Allow schema drift"**.
* Use **select()**, **derivedColumn()**, or **sink()** transformations with flexible column handling.

**2. Use Auto-Mapping in Copy Activity**

If the **column names and order match**, enable **Auto Mapping** in the **Copy Activity** sink.

* Automatically adjusts to minor schema changes like added columns.
* Simplifies pipeline maintenance.

**3. Use the Get Metadata Activity to Validate Schema**

* Use **Get Metadata** to fetch column schema (structure) from source.
* Use **If Condition** or **Stored Procedure** to compare with expected schema.
* Log or notify when mismatches occur.

**4. Use Parameterized Datasets & Pipelines**

Design your pipelines with **flexible datasets** using:

* **Dynamic file paths**
* **Dynamic table names or column lists**
* Use **@dataset().columnList** or **parameter passing** to adapt dynamically.

1. What's the purpose of a Self hosted Integration Runtime, and when would you use it?

The **Self-hosted Integration Runtime** is a component in ADF that allows **data movement** and **transformation** between:

* On-premises data sources
* Private network resources (e.g., a virtual machine, SQL Server in your office)
* Data stores that are **behind firewalls** or **not directly accessible from Azure**

You **install the SHIR** as a service on a machine (Windows OS).

It **authenticates with Azure Data Factory** using a **key**.

When a pipeline runs, ADF **instructs the SHIR** to move/transform data.

The SHIR does the work **locally**, and **communicates securely** with Azure.

🟢 **Azure IR**: Copy data between cloud sources like Blob, SQL DB, Snowflake.

🟠 **Self-hosted IR**: Access on-prem SQL Server, SFTP, or data behind firewalls.

🔵 **Azure-SSIS IR**: When migrating or executing SSIS packages.

1. How do you integrate ADF with Databricks or Synapse Analytics?

**🔗 1. Integration with Azure Databricks**

**✅ Use Cases:**

* Run **notebooks** for data transformation, machine learning, ETL.
* Process **large datasets** from Blob, ADLS, or external sources.

**🛠️ Steps:**

**➤ A. Create a Linked Service for Databricks**

* Go to ADF > **Manage** > **Linked Services**
* Create a new Linked Service → Choose **Azure Databricks**
* Provide:
  + **Workspace URL**
  + **Access token**
  + Choose **Cluster** or **Job cluster**

**➤ B. Use a Databricks Notebook Activity in a Pipeline**

* In your pipeline, add a **"Databricks Notebook"** activity.
* Select the linked service you created.
* Choose the notebook path and pass **parameters** (if any).

**✅ Example:**

Use Databricks to clean and aggregate data, then pass it to a Copy Activity to load into Azure SQL.

**🔗 2. Integration with Azure Synapse Analytics**

**✅ Use Cases:**

* Run **T-SQL scripts**, stored procedures
* Load and analyze **data warehouse-scale datasets**

**🛠️ Steps:**

**➤ A. Create a Linked Service to Synapse**

* Go to ADF > **Manage** > **Linked Services**
* Create new → Choose **Azure Synapse Analytics**
* Provide:
  + **Server name**
  + **Database name**
  + **Authentication method** (SQL Auth or Managed Identity)

**➤ B. Use Activities in Pipelines**

* **Stored Procedure Activity**: to call T-SQL stored procs in Synapse.
* **Lookup Activity**: to query tables/views.
* **Script Activity**: to run custom T-SQL inline code.

**➤ C. Copy Activity for Data Movement**

* Source from ADLS, Blob, etc.
* Sink to Synapse table using **PolyBase** or **Copy Command** for performance.

1. What are ARM templates, and how are they used in ADF?

**ARM (Azure Resource Manager) templates** are **JSON files** that define the **infrastructure and configuration** of Azure resources — including everything in **Azure Data Factory (ADF)**.

They are **infrastructure as code** tools that let you **automate deployments** of ADF resources like:

* Pipelines
* Datasets
* Linked Services
* Triggers
* Data Flows

They allow you to:

* **Export your entire ADF** setup (pipelines, datasets, etc.)
* **Deploy the same ADF configuration** across different environments (dev, test, prod)
* **Version control** your ADF code using Git
* Support **CI/CD** (Continuous Integration / Continuous Deployment)

1. Types of SCD  
   Slowly Changing Dimensions (SCD) are techniques used in data warehousing to manage **historical data** in dimension tables when data changes over time.

**SCD Type 0 – Fixed Dimension**

* **No changes allowed** after initial insert.
* Used for attributes that **should never change** (e.g., date of birth).

**SCD Type 1 – Overwrite**

* **Updates the existing record** with new values.
* **No history** is preserved.

**SCD Type 2 – History Preserved**

* Keeps **both old and new records** by adding a new row.
* Typically uses:
  + StartDate
  + EndDate
  + IsCurrent flag
  + Surrogate key

**SCD Type 3 – Partial History**

* Adds a **new column** for the previous value.
* Useful when **only recent change** is relevant.

**SCD Type 4 – History Table**

* Main table holds current data.
* Separate **history table** stores past changes.

1. Describe the steps for deploying ADF pipelines using CI/CD

**Step 1: Connect ADF to Git Repository**

* In ADF Studio:
  + Go to **Manage** > **Git Configuration**.
  + Connect to your **Azure DevOps repo**, select the project, repository, and collaboration branch (usually main or master).

**Step 2: Publish from Git to ADF**

* Once development is complete:
  + Click **Publish** in ADF Studio.
  + This creates an **ARM template** (ARMTemplateForFactory.json)

**Step 3: Create a CI/CD Pipeline in Azure DevOps**

* **Triggers** when code is pushed to a branch (e.g., main).
* **Extracts** the ARM templates for deployment.

**Step 4: Release Pipeline (CD)**

* Uses the artifact from the CI pipeline.
* Deploys to **test, staging, or production** ADF instance.

**Step 5: Parameterize for Each Environment**

* Use the ARMTemplateParametersForFactory.json to provide:
  + Environment-specific linked services (like Key Vault, Storage, SQL)
  + Dataset connections

1. What are Data Flows in Azure Data Factory?

**Data Flows** in Azure Data Factory are **visual, no-code data transformation** components that allow you to perform complex data manipulation at scale **without writing Spark code**.

They're executed on **Azure Data Factory’s managed Spark cluster**, so you get the power of distributed data processing with an easy drag-and-drop interface.

**Types of Data Flows**

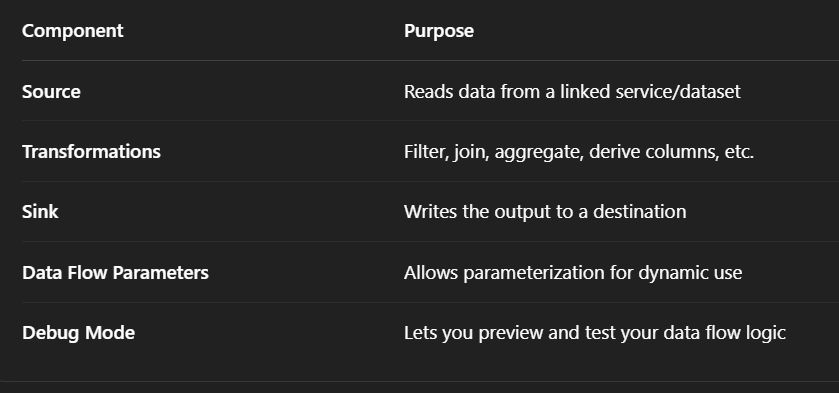
**1. Mapping Data Flows**

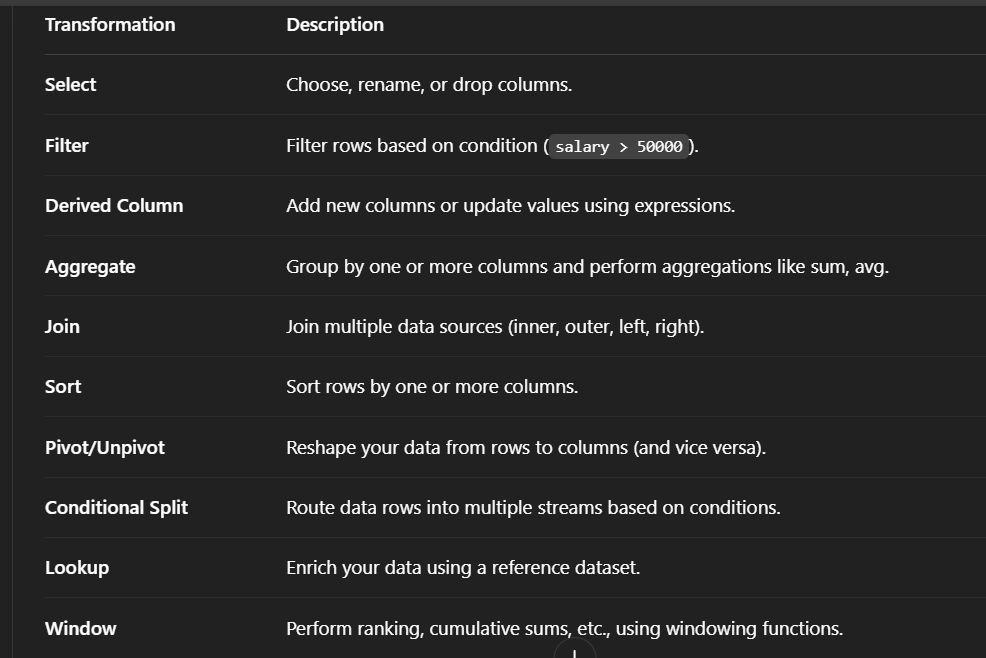
Used for **ETL (Extract, Transform, Load)** operations.

* Visual builder for designing data transformation logic.
* Supports joins, filters, aggregations, pivots, derived columns, lookups, etc.

**2. Wrangling Data Flows *(Less commonly used)***

* Based on Power Query (Excel-like interface).
* Best for data prep and exploration.





1. What is data lineage, and how would you show it in an Azure data pipeline?
   1. **Using ADF's Built-in Monitoring & Visual Tools**
   2. Enable Data Flow Debugging
   3. Integrate with Microsoft Purview
2. How do you manage access and security in Azure Data Lake Storage

**Authentication Methods**

* **Azure Active Directory (Azure AD)**:  
  Used to authenticate users, groups, and applications.
* **Shared Key**:  
  A storage account access key (less secure, mostly used for service access).
* **SAS (Shared Access Signature)**:  
  Time-limited, permission-scoped access via URL.

**Role-Based Access Control (RBAC)**

* Set at the **Azure resource level** (subscription, resource group, or storage account).
* Use **built-in roles** like:
  + Storage Blob Data Reader
  + Storage Blob Data Contributor
  + Storage Blob Data Owner
* RBAC defines **who can access the storage account** and perform high-level operations.

**Access Control Lists (ACLs)**

* Set at the **file system (container), folder, or file level**.
* Works similar to POSIX permissions:  
  Read (r), Write (w), Execute (x)
* ACLs provide **fine-grained data access control**.

**Network Security**

**Private Endpoints**

* Isolate storage from public internet access.
* Enable **VNet integration** for secure data pipelines.

**Firewall Rules**

* Whitelist specific **IP ranges** or **VNets**.

1. How would you improve the performance of data partitioning in ADLS?

**Partition by Relevant Columns**

* Use **columns with high cardinality** and those frequently used in filters or joins.
  + ✅ *Good:* year, month, region, product\_id

**Use Hierarchical Folder Structures**

ADLS Gen2 supports **Hierarchical Namespace**, which improves:

* Directory-level metadata
* Faster lookup & deletion
* Better integration with Hive-style partitioning

Example structure:  
sales/year=2024/month=03/day=10/\*.parquet

**Use Parquet or ORC Formats**

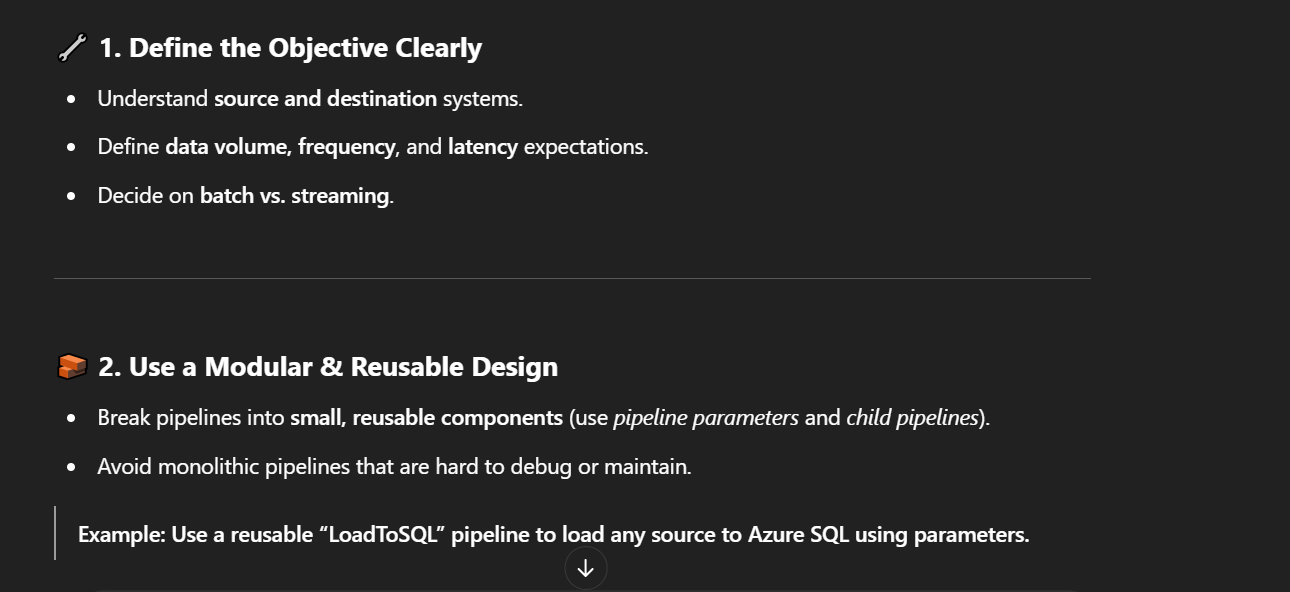
* These **columnar storage formats**:
  + Support partition pruning
  + Reduce I/O by only reading relevant columns
  + Work well with ADF, Synapse, and Databricks

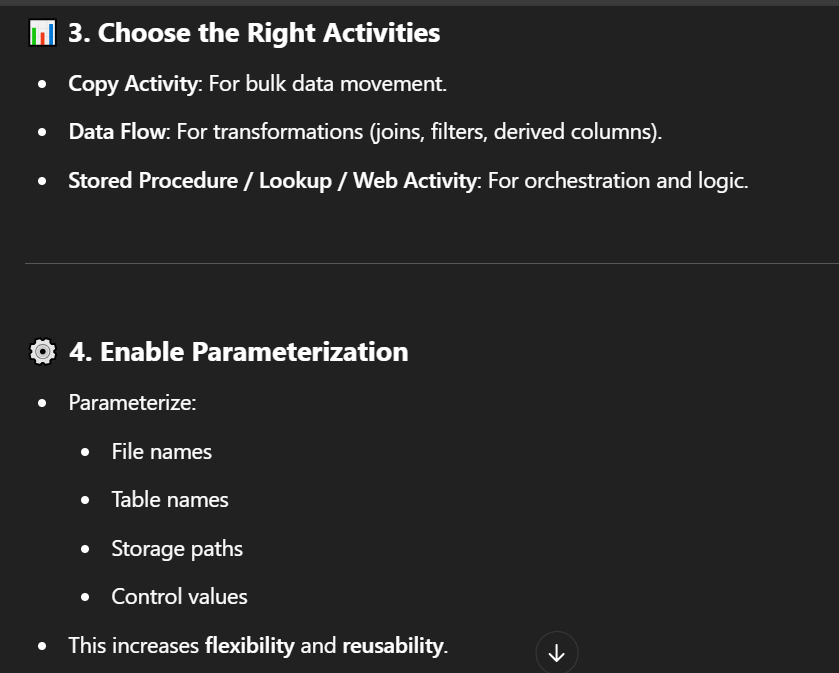
**Optimize File Size and Count**

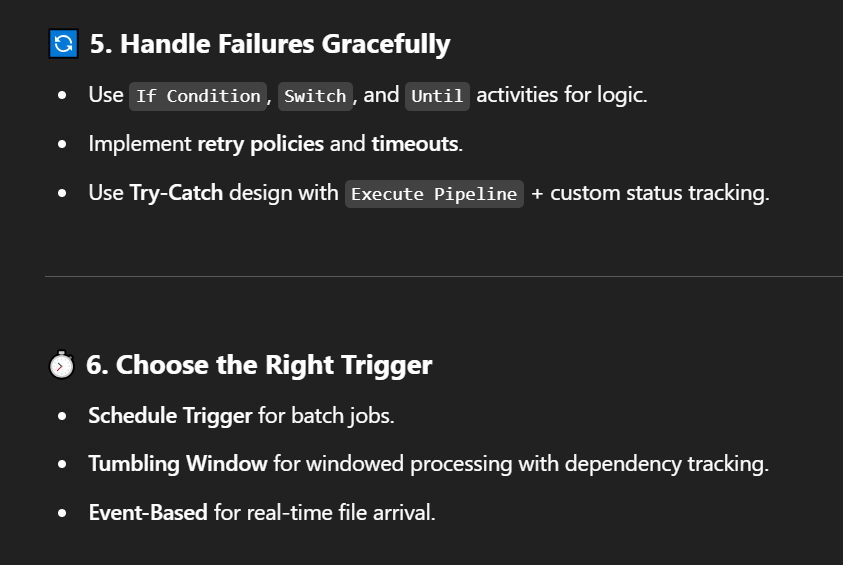
* Aim for **100–250 MB per file** for parallelism efficiency.
* Avoid:
  + **Too many small files** (slow listing, high metadata overhead)
  + **Very large files** (reduce parallelism)

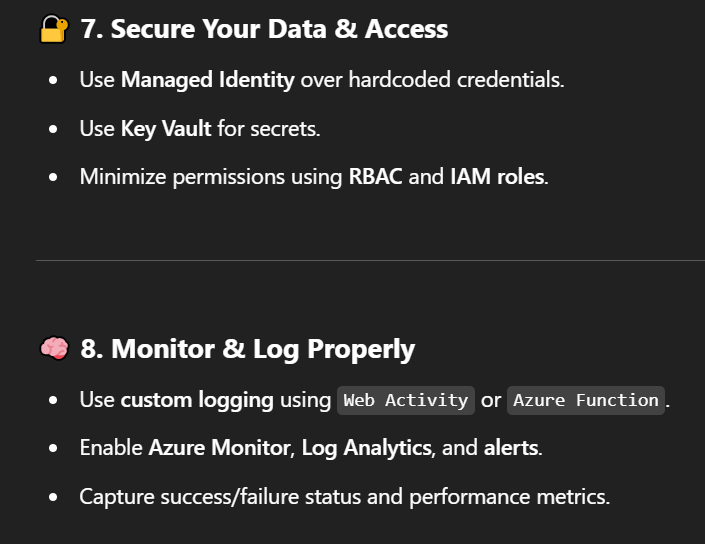
✔ Combine small files using tools like ADF Data Flows or Spark jobs.

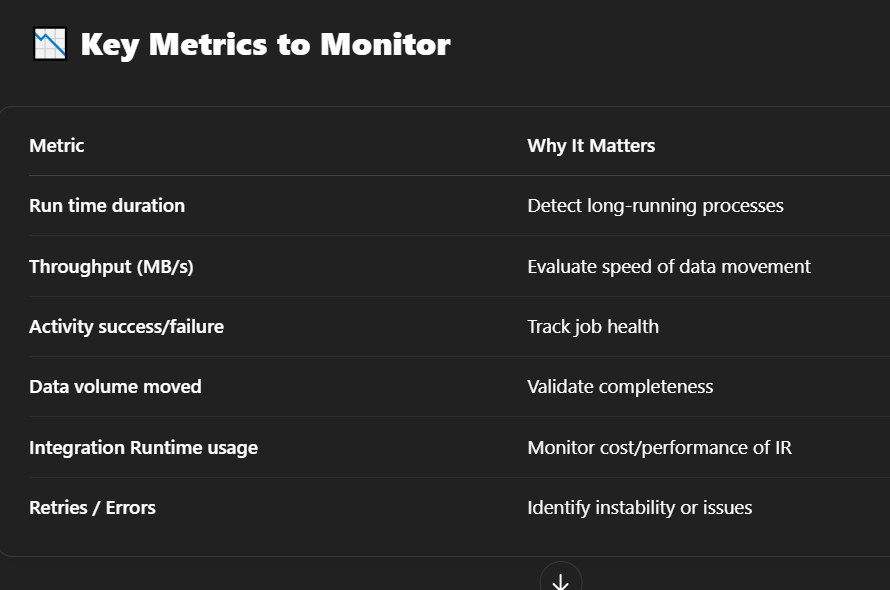
1. How do you design an effective ADF pipeline and what all metrics and considerations you should keep in mind while designing?

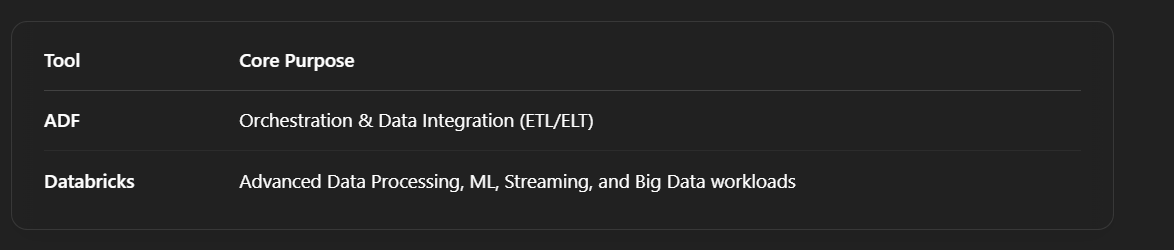






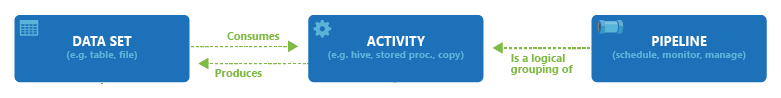






**What is Activity in Azure Data Factory?**

The activity is the task we performed on our data. We use activity inside the Azure Data Factory pipelines. ADF pipelines are a group of one or more activities. For ex: When you create an ADF pipeline to perform ETL you can use multiple activities to extract data, transform data and load data to your data warehouse. Activity uses Input and output datasets. Dataset represents your data if it is tables, files, folders etc. Below diagram shows the relationship between Activity, dataset and pipeline:



An Input dataset simply tells you about the input data and it’s schema. And an Output dataset will tell you about the output data and it’s schema. You can attach zero or more Input datasets and one or more Output datasets. Activities in Azure Data Factory can be broadly categorized as:

1- Data Movement Activities

2- Data Transformation Activities

3- Control Activities

**DATA MOVEMENT ACTIVITIES :**

**1- Copy Activity:** It simply copies the data from Source location to destination location. Azure supports multiple data store locations such as Azure Storage, Azure DBs, NoSQL, Files, etc.

To know more about Data Movement activities, please use below link:

**DATA TRANSFORMATION ACTIVITIES:**

**1- Data Flow:**In data flow, First, you need to design data transformation workflow to transform or move data. Then you can call Data Flow activity inside the ADF pipeline. It runs on Scaled out Apache Spark Clusters. There are two types of DataFlows: Mapping and Wrangling DataFlows

**MAPPING DATA FLOW:** It provides a platform to graphically design data transformation logic. You don’t need to write code. Once your data flow is complete, you can use it as an Activity in ADF pipelines.

**WRANGLING DATA FLOW:**It provides a platform to use power query in Azure Data Factory which is available on Ms excel. You can use power query M functions also on the cloud.

**2- Hive Activity:** This is a HD insight activity that executes Hive queries on windows/linux based HDInsight cluster. It is used to process and analyze structured data.

**3- Pig activity:** This is a HD insight activity that executes Pig queries on windows/linux based HDInsight cluster. It is used to analyse large datasets.

**4- MapReduce:**This is a HD insight activity that executes MapReduce programs on windows/linux based HDInsight cluster. It is used for processing and generating large datasets with a parallel distributed algorithm on a cluster.

**5- Hadoop Streaming:** This is a HD Insight activity that executes Hadoop streaming program on windows/linux based HDInsight cluster. It is used to write mappers and reducers with any executable script in any language like Python, C++ etc.

**6- Spark:**This is a HD Insight activity that executes Spark program on windows/linux based HDInsight cluster. It is used for large scale data processing.

**7- Stored Procedure:**In Data Factory pipeline, you can use execute Stored procedure activity to invoke a SQL Server Stored procedure. You can use the following data stores: Azure SQL Database, Azure Synapse Analytics, SQL Server Database, etc.

**8- U-SQL:** It executes U-SQL script on Azure Data Lake Analytics cluster. It is a big data query language that provides benefits of SQL.

**9- Custom Activity:** In custom activity, you can create your own data processing logic that is not provided by Azure. You can configure .Net activity or R activity that will run on Azure Batch service or an Azure HDInsight cluster.

**10- Databricks Notebook:**It runs your databricks notebook on Azure databricks workspace. It runs on Apache spark.

**11- Databricks Python Activity:** This activity will run your python files on Azure Databricks cluster.

**12- Azure Functions:** It is Azure Compute service that allows us to write code logic and use it based on events without installing any infrastructure. It stores your code into Storage and keep the logs in application Insights.Key points of Azure Functions are :

1- It is a Serverless service.

2- It has Multiple languages available : C#, Java, Javascript, Python and PowerShell

3- It is a Pay as you go Model.

To know more about Data Transformation activity, use below link:

**3- Control Flow Activities:**

**1- Append Variable Activity:**It assigns a value to the array variable.

**2- Execute Pipeline Activity:** It allows you to call Azure Data Factory pipelines.

**3- Filter Activity:** It allows you to apply different filters on your input dataset.

**4- For Each Activity:** It provides the functionality of a for each loop that executes for multiple iterations.

**5- Get Metadata Activity:**It is used to get metadata of files/folders. You need to provide the type of metadata you require: childItems, columnCount, contentMDS, exists, itemName, itemType, lastModified, size, structure, created etc.

**6- If condition Activity:** It provides the same functionality as If statement, it executes the set of expressions based on if the condition evaluates to true or false.

**7- Lookup Activity:** It reads and returns the content of multiple data sources such as files or tables or databases. It could also return the result set of a query or stored procedures.

**8- Set Variable Activity:** It is used to set the value to a variable of type String, Array, etc.

**9- Switch Activity:**It is a Switch statement that executes the set of activities based on matching cases.

**10- Until Activity:** It is same as do until loop. It executes a set of activities until the condition is set to true.

**11- Validation Activity:** It is used to validate the input dataset.

**12- Wait Activity:**It just waits for the given interval of time before moving ahead to the next activity. You can specify the number of seconds.

**13- Web Activity:** It is used to make a call to REST APIs. You can use it for different use cases such as ADF pipeline execution.

**14- Webhook Activity:** It is used to to call the endpoint URLs to start/stop the execution of the pipelines. You can call external URLs also.

Triggers :  
  
In **Azure Data Factory (ADF)** or **Synapse Pipelines**, **Tumbling Window Triggers** and **Schedule Triggers** are both used to initiate pipeline executions at specific times, but they differ significantly in behavior and use cases.

**Tumbling Window Trigger**

* **Interval-based and stateful**
* Executes pipelines in **fixed-size, non-overlapping time windows** (e.g., hourly, daily).
* **Waits for the previous window to complete** before starting the next.
* Supports **retry** and **dependency** tracking for each time window.
* Tracks metadata of execution windows — useful for **incremental loads**, **time series data processing**, etc.

**Example Use Case:**  
If you process log files every hour and want to ensure each hour’s data is processed exactly once, use tumbling window.

**Schedule Trigger**

* **Simple time-based (cron-like) trigger**
* Executes pipeline at a **specific time or recurring schedule**.
* **Stateless** — does not track what happened in the previous runs.
* No concept of windows or dependency tracking.

**Example Use Case:**  
If you want to run a pipeline **every day at 6 AM**, regardless of previous runs or data state, use a schedule trigger.

**Summary Table:**

| **Feature** | **Tumbling Window Trigger** | **Schedule Trigger** |
| --- | --- | --- |
| **Overlap/Gap Handling** | No overlap (fixed windows) | Can overlap (cron-like) |
| **State Tracking** | Yes (per window) | No |
| **Dependency on Previous** | Yes (waits for completion) | No |
| **Use Case** | Incremental loads, time series | Simple scheduled tasks |
| **Window Size** | Fixed intervals (e.g., 15 min) | Customizable with cron |
| **Supports Retry** | Yes | Limited |

1. There are 10 million CSV files in ADLS Gen2. How will you read them and process using Azure Data Factory? Explain in detail what and all would be used.

**1. Use Metadata-Driven File Processing**

Instead of scanning ADLS every time, maintain a **metadata table** that tracks:

* File name
* File path
* Processing status (NotStarted, InProgress, Completed, Failed)
* Timestamps

You can store this in an **Azure SQL Database** or **Azure Synapse SQL Pool**.

**2. Create a Parameterized Pipeline**

Create a single ADF pipeline that can process one file at a time using **parameterized datasets and pipeline parameters**:

* @pipeline().parameters.filePath
* @pipeline().parameters.fileName

This allows reuse of the same pipeline across millions of files dynamically.

**3. Use Lookup + ForEach Pattern (Batch Mode)**

Instead of trying to process 10 million files at once:

* **Stage 1 – Metadata Initialization**:
  + Use a **Data Flow or custom script** to list all files in the ADLS Gen2 path using **Get Metadata** activity with "childItems".
  + Load this list into the metadata table.
* **Stage 2 – Batch Processing**:
  + Use a **Lookup activity** to read a batch of, say, 1000 unprocessed file paths from the metadata table.
  + Use a **ForEach activity** (in batch mode with batchCount) to process the files in parallel using the parameterized pipeline.

**4. Copy Activity or Data Flow to Process Files**

Inside the processing pipeline:

* Use a **Copy Activity** or **Mapping Data Flow** to:
  + Read the CSV file from ADLS Gen2
  + Transform it if needed (cleaning, joining, etc.)
  + Write the output to a target (SQL, Synapse, another ADLS path)
* On success/failure, **update the metadata table** using a **Stored Procedure Activity** or **Web Activity + REST API**.

**5. Monitoring and Retry Mechanism**

* Use **dependency tracking** and **logging** to track which files succeeded/failed.
* Build a **retry mechanism**:
  + Filter only failed files in the next run.
  + Reprocess them with updated logic or resources.

**6. Optimize Performance**

To handle large scale:

* Enable **Parallel ForEach** (up to 50 concurrent executions).
* Use **Integration Runtimes with scaling enabled**.
* Use **partitioning** in Copy Activity/Data Flow when applicable (e.g., column-based partitioning).
* **Avoid listing 10M files every time** – rely on metadata.

**7. Automation & Scheduling**

Use **Tumbling Window Trigger** or **Schedule Trigger** to run the pipeline in scheduled batches:

* Every hour, fetch the next batch of unprocessed files.
* Use **self-dependency** if you want to avoid overlapping execution.  
    
    
  Basically explain DCF

1. What is variable and parameter in ADF?

| **Feature** | **Parameter** | **Variable** |
| --- | --- | --- |
| **Scope** | Passed into pipeline or dataset (read-only) | Defined and used inside a pipeline (read/write) |
| **Mutability** | **Immutable** during execution | **Mutable** – can be set or updated dynamically |
| **Use Case** | For configuring pipeline at run-time | For storing temporary state or counters |
| **Data Type** | String, Int, Bool, Array, Object | String, Int, Bool, Array |

1. The pipeline is scheduled and it got failed. I need to automate it by sending the mail when it gets failed. How would you implement it?

* Use a **Web Activity** to call a Logic App or email API in the **failure path** of your pipeline.

**Steps:**

1. Create a **Logic App** that receives a POST request and sends an email.
2. In ADF, use **Web Activity** in the **On Failure** path of your pipeline.
3. Pass pipeline name, run ID, and error details in the POST body.

**Use Azure Monitor + Logic App (Recommended for Production)**

**Step-by-Step Setup**

**Step 1: Enable ADF Diagnostics Logs**

1. Go to your **Data Factory** in Azure portal.
2. Under **Monitoring → Diagnostic settings**, click **+ Add diagnostic setting**.
3. Enable:
   * **PipelineRuns**
   * **ActivityRuns**
4. Send diagnostics to **Log Analytics workspace**.

**Step 2: Create a Logic App to Send Email**

1. Go to Azure Portal → **Create a Logic App (Consumption)**.
2. In the Logic App Designer:
   * Trigger: **When a resource event occurs**
     + Subscription: your Azure subscription
     + Resource type: Microsoft.DataFactory/factories
     + Event type: Microsoft.DataFactory.PipelineRunFailed
   * Action: **Send email** (use Outlook, Office 365, or SMTP)
     + Fill in To, Subject, Body with dynamic pipeline info.

**Step 3: Test the Setup**

* Run a pipeline with intentional failure to trigger the email.
* You will receive an email with details like pipeline name, run ID, error message.

1. How do you handle the exceptions in ADF?

**1. Activity Dependency Conditions (On Failure, On Completion, On Skip)**

You can branch activities based on their status:

* **On Success** → default behavior
* **On Failure** → run an error handler (e.g., send email, log error)
* **On Completion** → run cleanup (always runs regardless of success/failure)
* **On Skip** → if activity is skipped

🧠 **Use case**: Run a Web/Stored Procedure activity to log the failure or send an email.

**2. Try-Catch Pattern using If Condition or Switch**

You can simulate a try-catch structure:

* Main pipeline logic in one sequence
* Failure path uses:
  + **Set Variable** or **Execute Pipeline**
  + **Web Activity** to trigger notification
  + **Stored Procedure** to log exception in DB

**3. Use Fail Activity to Raise Custom Errors**

* You can forcefully fail a pipeline using a **Fail activity** with a custom message.

json

CopyEdit

{

"type": "Fail",

"typeProperties": {

"message": "Invalid file format detected"

}

}

**4. Logging Errors into a Table**

* Use **Stored Procedure Activity** to write the failure details (pipeline name, activity name, error message, timestamp) into a SQL table for auditing and reporting.

**5. Retry Policies**

Each activity (e.g., Copy Activity) supports:

* **Retry count**
* **Retry interval (in seconds)**

🔁 Example:

json

CopyEdit

"policy": {

"retry": 3,

"timeout": "00:10:00",

"retryIntervalInSeconds": 60

}

Use this for **transient errors** like network or service timeouts.

**6. Global Error Handling via Execute Pipeline**

You can centralize error handling:

* In your main pipeline, call a child pipeline with On Failure.
* The child pipeline handles:
  + Logging to DB
  + Sending alert email
  + Archiving failed file

1. If the ADF pipeline is running very slow, how would you approach and fix it?

**✅ 1. Check Pipeline Monitoring and Activity Duration**

* Go to **Monitor tab** in ADF.
* Check:
  + Which activity is taking the most time (e.g., Copy, Data Flow, Lookup).
  + Pipeline run history — is it consistently slow or only for specific runs?

**2. If It's a Copy Activity – Optimize This Way**

**a. Enable Staging (for large data movement)**

* Use **Blob or ADLS staging** when copying between cloud sources/sinks (like SQL to Snowflake).
* Improves performance for connectors that support it.

**b. Use Parallelism / Partitioning**

* Set Degree of copy parallelism.
* Use **source partitioning** (e.g., based on a column like date or ID).
* Example: Divide source data by year or month.

**c. Optimize Source Query**

* Use **query pushdown** (SQL query that filters early).
* Avoid pulling more columns/rows than needed.
* Avoid SELECT \*.

**d. Use Proper Integration Runtime (IR)**

* Use **Azure IR** for cloud-to-cloud.
* Use **Self-hosted IR** for on-prem.
* Ensure **IR is not overloaded** or throttled.

**3. If It's a Data Flow – Tune Spark Execution**

**a. Set Compute Size Appropriately**

* Use higher **core count** for complex transformations.

**b. Reduce Dataset Size**

* Filter data as early as possible.
* Cache reusable datasets.

**c. Optimize Joins**

* Broadcast small dataset if possible.
* Ensure join keys are indexed or sorted.

**d. Monitor Data Flow Execution**

* Use **Data Flow Debug mode** and **monitor execution time** per transformation.

**4. Use Concurrency & Parallel Activities**

* Split into **parallel branches** using If Condition or ForEach with Batch Count.
* Example: process multiple files in parallel from a folder.

**5. Tune Integration Runtime Settings**

* Check the IR capacity:
  + Number of nodes
  + CPU usage
  + Memory
* Consider **scaling up Self-hosted IR** if it’s bottlenecked.

1. What is the difference between Blob storage and ADLS Gen2? Why ADLS Gen2 is required?

**Difference Between Blob Storage and ADLS Gen2**

Both **Azure Blob Storage** and **Azure Data Lake Storage Gen2 (ADLS Gen2)** are used to store large amounts of unstructured data in Azure, but they have key differences in terms of **features, capabilities**, and intended use cases.

| **Feature** | **Blob Storage** | **ADLS Gen2** |
| --- | --- | --- |
| **Base Storage Type** | **Blob Storage** (Block blobs) | **Data Lake Storage Gen2** (Built on Blob Storage) |
| **Hierarchical Namespace** | **No** – Flat namespace | **Yes** – Hierarchical (Folder-like structure) |
| **Azure Active Directory (AAD) Authentication** | **Limited support** (only for Azure Blob Storage Access via Shared Access Signatures or keys) | **Full support** – Supports Azure AD-based access control (RBAC) |
| **ACLs (Access Control Lists)** | **No** | **Yes** – Fine-grained security using ACLs on directories and files |
| **Performance** | High-performance object storage for large binary and text files | Optimized for big data workloads, enabling faster read and write operations |
| **Optimized for** | Object storage (general use) | Big data analytics workloads (e.g., Hadoop, Spark) |
| **Data Lake Specific Features** | Not present | Built-in support for large-scale analytics and processing tools |
| **Cost** | Typically lower for simple use cases | Higher cost for advanced features but optimized for analytics |
| **Use Case** | General-purpose storage (e.g., web apps, backups) | Data Lakes, Big Data Analytics, ETL workflows, ML |

**Why ADLS Gen2 is Required**

1. **Hierarchical Namespace for Folder Structure**:
   * **ADLS Gen2** supports a **hierarchical namespace** (similar to a file system) which allows data to be stored in directories and subdirectories. This is essential for managing and organizing large datasets in big data and analytics workloads.
   * In contrast, **Blob Storage** uses a **flat namespace**, where all blobs are treated as flat objects, which is less efficient for managing hierarchical data structures.
2. **Fine-grained Access Control (ACLs)**:
   * **ADLS Gen2** allows the use of **Access Control Lists (ACLs)** to provide more granular and fine-tuned access control on individual files or directories. This is vital for large data lake environments where different users or groups require different access levels.
   * **Blob Storage** does not natively support this level of fine-grained access control, relying instead on SAS tokens or shared keys.
3. **Optimized for Big Data Analytics**:
   * **ADLS Gen2** is specifically designed to integrate seamlessly with big data analytics services like **Azure Databricks**, **HDInsight**, **Azure Synapse Analytics**, and **Azure Data Factory**.
   * It supports **Hadoop Distributed File System (HDFS)** APIs, which makes it an ideal choice for analytics workloads that rely on these tools.
4. **Better Performance for Large Files**:
   * **ADLS Gen2** provides better performance for big data workloads due to optimizations for handling large files and high-throughput operations.
   * The ability to perform **parallel read/write operations** makes it more efficient for processing large datasets in analytics pipelines.
5. **Azure Active Directory (AAD) Integration**:
   * **ADLS Gen2** integrates with **Azure Active Directory (AAD)** for user authentication and role-based access control (RBAC), allowing you to define security policies and access at a user/group level, which is critical for compliance and security in enterprise environments.
   * **Blob Storage** has limited integration with AAD, making it harder to manage granular access for large teams or organizations.
6. **Cost-Effectiveness for Big Data**:
   * While **Blob Storage** might be cheaper for general-purpose use, **ADLS Gen2** offers better cost optimization for **analytics workloads**, especially with large amounts of data. The cost of storing data is optimized for scenarios like **data lakes** and **batch analytics** where files can be organized and processed efficiently.
7. Discuss key concepts in data modeling, including normalization and denormalization.

**Data modeling** involves designing the structure of data, how it is stored, and how it will be queried. It ensures the data is organized efficiently and is easy to query while maintaining integrity, consistency, and performance. Two important concepts in data modeling are **Normalization** and **Denormalization**. Let's explore them:

**1. Normalization**

**Normalization** is the process of organizing data to reduce redundancy and improve data integrity. It involves dividing large tables into smaller ones and defining relationships between them. The goal is to store data in such a way that it minimizes duplication and ensures consistency.

**Key Goals of Normalization:**

* **Reduce Data Redundancy**: Avoid storing the same data multiple times.
* **Minimize Data Anomalies**: Ensure that operations like insert, update, and delete can be performed consistently.
* **Ensure Data Integrity**: Ensure relationships between entities are maintained (using primary and foreign keys).

**Common Normal Forms:**

1. **1NF (First Normal Form)**: Remove duplicate records, and ensure each field contains atomic values (no multiple values in one field).
2. **2NF (Second Normal Form)**: Achieve 1NF and remove partial dependencies (non-key attributes must depend on the entire primary key).
3. **3NF (Third Normal Form)**: Achieve 2NF and remove transitive dependencies (non-key attributes should not depend on other non-key attributes).

**2. Denormalization**

**Denormalization** is the process of combining normalized tables or adding redundant data to reduce the number of joins needed during queries. It can improve query performance in data warehouses by making data retrieval faster but at the cost of data redundancy and potential inconsistencies.

**When to Use Denormalization:**

* **Performance**: If data is frequently queried with complex joins, denormalization reduces the number of joins.
* **Query Speed**: Denormalization is commonly used in **Data Warehouses** where reporting speed is a priority.
* **ETL Pipelines**: In data transformation scenarios, it may be beneficial to denormalize the data to make processing faster.

1. How can you mask sensitive data in the Azure SQL Database? What are the different masking techniques available?

**Masking Techniques Available in Azure SQL Database**

Azure SQL Database offers several masking techniques for different types of data. These techniques are applied when you define the mask on specific columns in a table.

**1. Default Masking**

The **default mask** is a built-in masking rule that automatically masks the data based on its data type.

* **For numeric values**: It masks the value by replacing the actual number with a range, such as 0.
* **For date/time values**: It replaces the actual date or time with a placeholder value like 1900-01-01.

**Example**:

sql

CopyEdit

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FullName NVARCHAR(100) MASKED WITH (FUNCTION = 'default()'),

DateOfBirth DATE MASKED WITH (FUNCTION = 'default()')

);

Here, the **DateOfBirth** and **FullName** columns are automatically masked.

**2. Email Masking**

For email addresses, Azure SQL Database has a built-in email mask function that obfuscates email addresses, but still shows the domain.

* **Example**:  
  For user@example.com, it would display as xxxx@example.com.

**Example**:

CREATE TABLE Users (

UserID INT PRIMARY KEY,

Email NVARCHAR(100) MASKED WITH (FUNCTION = 'email()')

);

**3. Random Masking**

The **random mask** replaces actual data with a random value within a defined range. This is useful for scenarios where you want to hide the exact value but still provide a realistic value for applications or reports.

* For **numeric** data types, the random mask provides a random number within a defined range (e.g., 1000 to 9999).
* For **date/time** data types, it provides a random date within a specified range.

**Example**:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Salary INT MASKED WITH (FUNCTION = 'random(1000,9999)')

);

Here, the Salary column will return random values between 1000 and 9999.

**4. Custom Masking**

You can define custom masking rules using **functions** for your specific needs. For example:

* A **string** can be partially masked, revealing only the last few characters or the first few characters.
* For **numeric** data types, you could define a range or apply custom logic.

**Example (Partial String Masking)**:

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CreditCardNumber NVARCHAR(16) MASKED WITH (FUNCTION = 'partial(4,"xxxx",4)')

);

In this example, the CreditCardNumber column is partially masked, and it will show only the first 4 and last 4 digits, with xxxx in between.

**Explanation**:  
partial(start, middle, end) function works as follows:

* **start**: Number of characters to reveal at the start of the string (4 in this case).
* **middle**: The text that will be masked in the middle (e.g., xxxx).
* **end**: Number of characters to reveal at the end of the string (4 in this case).

**5. Null Masking**

This masking technique returns NULL for all data in the column for users without the necessary permissions to see the data.

**Example**:

CREATE TABLE Financials (

TransactionID INT PRIMARY KEY,

AccountNumber NVARCHAR(20) MASKED WITH (FUNCTION = 'null()')

);

This will cause the AccountNumber column to return NULL when accessed by a non-privileged user.

**Applying Dynamic Data Masking (DDM)**

You can define dynamic data masking on a table column when creating the table or alter existing columns to apply the mask. Here's how to do it:

1. **When Creating a Table**:  
   You can directly apply masking in the column definition during table creation.

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name NVARCHAR(100) MASKED WITH (FUNCTION = 'default()'),

SSN NVARCHAR(11) MASKED WITH (FUNCTION = 'partial(3, "XXX-XX", 4)')

);

1. **Modifying an Existing Table**:  
   You can add or change masking on an existing column by using the ALTER TABLE statement.

sql

CopyEdit

ALTER TABLE Employees

ALTER COLUMN SSN ADD MASKED WITH (FUNCTION = 'default()');