1. What is Azure Data Factory?

- **Azure Data Factory (ADF)** is a cloud based **ETL** service provided and managed by Microsoft on its Azure cloud computing platform.
- It was released to public in **2015**.
- ADF automates data loads by providing a number of activities specific to tasks.
- It has 90+ connectors which ease the process of extracting or loading the data into source or target respectively.
- It offers mapping data flows which ease the tasks of transforming data seamlessly.
- It offers different integration runtimes namely Azure IR, Self-Hosted IR, Azure – SSIS IR to connect to different services or applications in Azure, on-premises and SSIS package migration respectively.
- ADF can be easily integration with Azure dev-ops for code repository, code versioning and deployments.

2. What are building blocks of Azure Data Factory?

- **Integration Runtime (IR):** Integration Runtime is the compute responsible for data movement and other activities. There are different integration runtimes namely Azure IR, Self-Hosted IR, Azure – SSIS IR to connect to different services or applications in Azure, on-premises and SSIS package migration respectively. Azure IR is fully managed by Microsoft and always up and running. We need to install the self-hosted IR and manage it.
- **Pipeline:** Pipeline is a logical flow of tasks. It is the top most level object in data factory. A pipeline can contain one or more activities which perform the tasks individually. These activities can run in serial or parallel depending on use case. You can create variables and parameters in pipeline to make your solution dynamic. You can also monitor the progress of pipeline run from the output tab. Pipelines can be executed to run on demand or by assigning triggers to them. Pipelines are of 2 types: Master and Child. Master pipeline can run independently whereas child pipelines are invoked from master pipeline. Kindly note, child pipelines can also run independently if you are not expecting any input from master pipeline.
- **Dataset:** A dataset is a reference or pointer to a service, application or tool which you want to connect to pull or push the data. A dataset can be consumed in multiple pipelines provided by don't change anything in it once you create it, which makes the dataset reusable or portable. Hence, the dataset is not tied to a specific pipeline.
- **Linked Service:** Linked service is an authentication mechanism which saves the credential details to connect to a service. It is consumed in a dataset. The same linked service can be used in n number of datasets provided you want to connect to same service. For instance.. a linked service created to connect to an Azure SQL database in a specific resource group can be reused multiple times in different datasets

when you want to connect to same database and likewise for other services. The connection has to succeed in order to interact with the service else the task will fail.

- **Activity:** To perform tasks there are different activities in ADF like copy data, for each, getmetadata, if, switch, etc. Each activity is used for specific task. Activities are added in a pipeline. The activities can be chained in serial or made to run in parallel depending on use case. The most used activities are copydata, foreach, if.
- **Mapping data flow:** Dataflow facilitates data transformations seamlessly. Without the need to write the code you can perform most of the data transformations easily just by configuring the settings.
- **Trigger:** Trigger automates execution of a pipeline with no manual intervention needed to run ADF pipelines. You have different types of triggers like schedule trigger, storage event trigger, custom event trigger and tumbling window trigger.
- **Parameter:** Parameter is defined on a pipeline level only. It makes the pipeline dynamic to accept input which can be used further in the pipeline execution. You can create max 50 parameters in a pipeline. For updates check the link: https://learn.microsoft.com/en-us/fabric/data-factory/data-factory-limitations . Parameter is read only during the execution of pipeline, you cannot change its value once the pipeline starts running.
- **Variable:** Variable is a container that can hold any value. You can create variables on a pipeline level and use it anywhere within it. Variables help you to avoid using the same value or expression repeatedly. You can change the value of a variable by using set variable activity.

3. What are different use cases of copy data activity?

- Copy data activity is used to copy data from source to destination. The source and destination can be any service, application or tool in Azure, on-premises on other cloud platform or independent tool.
- Copy data has 90+ connectors to connect to different services or tools which makes it easy to copy the data.
- The settings needed to be configured change according to the service selected in a dataset on the source or target side. For instance. When you connect to ADLS gen2, the options given will be different as compared to Azure SQL database.
- When you connect to ADLS gen2, you have a number of options to extract data from a specific file, folder, pattern based filter or list of files. You can also filter files modified between specific timeframe. If the data is partitioned, it can automatically discover the partitions and add them as additional columns in your data from source side by eliminating the need to manually discover the partitions.
- When you connect to Azure SQL database, you have options to extract an entire table, write a SQL query or run a stored procedure to extract data.
- You can also add additional columns when you read the data from source typically you can add info like filepath, filename, current date time, etc. in these columns which can be copied into sink.
- You can use the column mappings when the column names are not same on source and sink side. Here you can do a one to one mapping. If needed you can also make the column mappings by adding a dynamic expression in it.
- You can increase the degree of parallelism to speed up the copy, max can be set is 32.
- You can use logging to capture logs of copy task.

4. What is use of set variable activity?

- Set variable activity is used to assign a value to a variable. You can assign different values to a same variable in a pipeline each time by using set variable activity, but only the latest value will be retained in a variable.
- Set variable activity can also be used to save information when the expression is dynamic and you cannot track the details during the execution of pipeline.
- Variable created in one pipeline cannot be accessed in another pipeline directly as the scope of variable is only within the pipeline created. In case you want to access a value of variable created in one pipeline into another, you can create a parameter in another dependent pipeline and then pass the value of variable from earlier pipeline which has the variable.

5. What is use of get metadata activity?

- Get metadata activity is used to fetch metadata (data about data) or information about an object like folder in ADLS or table in database.
- It is mostly used to fetch metadata of a file or folder in ADLS which can return different properties like item name(file or folder name), size of file, contents of a folder(subfolder and file names one level under) , last modified date time for a file or folder, file or a folder existence, column count, etc. Every single property can be used for a specific scenario.
- The output of childitems property is an array, which can be used further in ForEach to extract item one by one and perform operations.

6. What is use of ForEach Activity?

- ForEach Activity is a loop task, it loops (iterates) over an array or group of items and returns them. The items can be returned serially or randomly. You can fetch the items one by one or in batches.
- ForEach has a built in expression **@item** which returns the current item returned by it.
- When the input is a simple array like [1,2,3,4,5] @item will simply return each item per iteration like 1, then 2, then 3 and so on.., but when the input is an object which happens in case you have a complex array returning child items of a folder or rows from a table, then in such case you cannot just write **@item,** it will fail stating cannot set object , so you need to add the property or column name like **@item.name** to refer to file name or **@item.columnname** for specific column value.
- The limitation of this activity is you cannot have another ForEach activity inside it, which means no nested ForEach is allowed, you will have to create another child pipeline and pass the parameter from master pipeline to overcome this limitation.
- ForEach activity is used to not create dynamic pipelines but also it reduces lengthy development. For instance.if you want to copy 3 different files, you need not add 3 separate copy data activities, instead one for each activity can do the job for you with only one copy data activity inside it and using the current item of for each and parameters you could make it dynamic.

7. What is truncate and load?

- Truncate and load is one of the data load types in ETL.
- In this process we delete or clear the entire existing data mostly stored in a table.
- Usually we follow this approach to make sure if any existing data is updated it can be replaced and if any new data is coming up that can be loaded as well.
- The drawbacks in this approach is this doesn't suit for large data loads, no matter how large compute is deployed for the task as you end up copying the entire data which mostly is unchanged, the time taken and costs will be huge and not beneficial for the business to get insights from the data in short time.

8. What is Upsert?

- Upsert is another type of data load in ETL.
- It is a combination of insert and update, which means update the existing data and insert the data if there are any new rows.
- In order to identify if the row exists or not it needs a key value which is unique like a primary key, based on this it can check whether the value exists or not and then updates or inserts the rows respectively.
- You can also combine more than one column for key in Upsert.
- This feature is available on sink side in data factory.

9. What is append variable activity?

- Append variable activity add a value and preserves the earlier values, this is biggest difference between set variable and append variable activity.
- Only array type variables can be used to append values as array can store multiple values.
- The use cases of this activity are rare.

10. What is delete activity?

- Delete activity is used to delete objects like files and folder.
- This is only compatible with storage services like ADLS, amazon S3, google cloud, etc.
- You can also recursively delete folders which means all the subfolders and files can be deleted by just ticking the recursively check box.
- You can delete files modified between specific timeframes as well by using the start and end date.
- You can log the details in ADLS for audit purpose.

11. What are differences between parameter and variable?

- Both parameter and variable are created on a pipeline level but there are few differences.
- You can control the value of a parameter before the execution of a pipeline starts, once the pipeline begins execution you cannot change the value of a parameter, however with variable it is not the case, you can keep on changing the value of a variable during the execution of pipeline using set variable activity multiple times.
- Parameter supports more datatypes as compared to variables.
- When you need to pass values from one pipeline to another parameter can be used but not variable.

12. What is Lookup activity?

- Lookup activity is used to fetch the contents of a file or table, in short it reads the data from a file or table and returns an array.
- It can return up to 5000 rows, in case result set contains more than 5000 rows it returns the top 5000 rows. The need to fetch more than 5000 rows is rare.
- It supports output of up to 4mb in size, if exceeded the activity fails.
- The output returned is in form of an array.
- By default it returns only first row, you can untick the checkbox **First row only** to return multiple rows.

13. What is If activity?

- If activity is used for conditional execution.
- It has a true part and false part.
- The diversion to true and false part happens based on logical expression which should yield either true or false. This can mostly be achieved using comparison functions like greater, equals, and, or, etc or any value which returns true or false readily.
- False part is optional which means there is no compulsion to add an activity in false part.
- The limitation of if activity is it cannot have another If activity inside it called as nested if.
- Another limitation is you cannot have ForEach activity inside it.
- To overcome nested if you have to use another If activity and re-evaluate the expression.
- To overcome usage of ForEach activity which cannot be added inside it, you can add an Execute pipeline activity and call a child pipeline which has ForEach activity in it.

14. What is switch activity?

- Switch activity switches to different conditions based on a value given in expression, this value can be any value and not specific to Boolean.
- It has a default case (condition) and you can add multiple cases, maximum up to 25 cases.

15. What are differences between If and switch activity?

- If needs an expression resulting in true or false, but switch can evaluate any value and doesn't restrict to only true or false.
- If has a true and false part, whereas switch has cases and it is easy to manage when you have multiple cases, which is lengthy with if activity as single if can't do it.

16. What is a stored procedure activity?

- Stored procedure activity is used to run stored procedure already created in a database.
- The stored procedure can be available in Azure SQL database, SQL server or Azure synapse analytics.
- In case the stored procedures need parameters you can import the parameter names in the activity and pass the values in them as needed.

17. What is script activity?

- Script activity is used to run DML and DDL statements directly from data factory.
- It can contain one or multiple SQL statements which run in sequential.
- You can create, alter, drop objects like tables and views.
- You can also run stored procedures from it.

18. What are different ways to execute a stored procedure in ADF?

- Copy data activity
- Lookup activity
- Stored procedure activity
- Script activity

19. What is Execute pipeline activity?

- Execute pipeline activity is used to run another pipeline.
- One execute pipeline activity can run only one pipeline, to run multiple pipelines from one pipeline you need to use as many execute pipeline activities.
- In case the child pipeline has parameters defined, you need to pass them in execute pipeline activity from the master pipeline.

20. What are different output modes for an activity?

- **On Skip:** In case the activity is **skipped** during execution it will run the next activity chained to it.
- **On Success :** In case the activity is **successful** during execution it will run the next activity chained to it
- **On Fail :** In case the activity is **failed** during execution it will run the next activity chained to it
- **On Completion:** In case the activity has **completed** execution **irrespective** of **success** or **failure** during execution it will run the next activity chained to it.

21. What is an Integration Runtime and list its types?

- Integration Runtime (IR) is the compute infrastructure used to move and transform data.
- Types of IR:
    - Azure IR: Cloud to Cloud
    - Self-hosted IR: On premises
    - Azure SSIS IR: Run SSIS packages

22. What are different types of triggers?

- **Schedule Trigger** – Time-based
- **Event Trigger** – Blob creation/deletion
- **Tumbling Window Trigger** – Fixed, incremental windows

23. How do you optimize Copy Activity performance?

- Use partitioning
- Enable parallel copy
- Use staging
- Choose correct IR
- Avoid small files

24. What is difference between dataset level and pipeline level parameters?

- Dataset level parameters are accessible only within the dataset, not outside the dataset, whereas pipeline level parameters are available in entire pipeline.
- Dataset level parameters are used to make dynamic folder path, table name and file names or entities, whereas pipeline level parameters can used to pass the values anywhere in throughout the pipeline which can used to make dynamic folder path, table name and file names or entities and expressions.

25. What are Global Parameters?

- Reusable parameters defined **once** at factory level, available across pipelines.

26. Can ADF process real-time data?

- **No — ADF is batch-**oriented

27. What are metadata driven pipelines?

- A metadata-driven ADF pipeline is an Azure Data Factory design pattern where the pipeline logic is generic, and all the behaviour (what to copy, from where, to where, how, and when) is controlled by metadata stored outside the pipeline—usually in tables or config files.
- In short 👇
  👉 Pipelines don't change. Metadata changes.

◇ What is "Metadata" here?

Metadata is **configuration data**, not business data.

Typically stored in:

- Azure SQL Database
- Azure Table Storage
- ADLS (JSON / CSV)
- Dataverse

Example metadata fields

Field	Meaning
source_type	sql / blob / adls
source_path	container/folder/table
target_path	target container/folder
file_format	csv / parquet / json
watermark_column	last_modified_date
load_type	full / incremental
is_active	Y / N

🔹 How a Metadata-Driven Pipeline Works (Flow)

1. **Lookup activity** reads metadata table
2. **ForEach activity** loops through metadata rows
3. **Dynamic expressions** configure:
   - Dataset
   - Source
   - Sink
   - File paths
   - Queries
4. **Single pipeline** processes **N tables/files**

📌 You add a **new source** → just insert a row in metadata
📌 No pipeline duplication
📌 No redeployment

🔹 **Why Metadata-Driven Pipelines Are Powerful**

✅ Highly **scalable**
✅ **Low maintenance**
✅ Enterprise-ready design
✅ Perfect for **Data Lake ingestion frameworks**
✅ CI/CD friendly
✅ Required skill for **ADF interviews**

🔹 **Metadata-Driven vs Traditional ADF**

Traditional ADF	Metadata-Driven ADF
One pipeline per table	One pipeline for all tables
Manual changes	Config-only changes
Hard-coded paths	Dynamic expressions
Not scalable	Enterprise scalable

🔹 **Real-World Use Cases**

- Bronze → Silver ingestion framework
- Dataverse / SAP / Oracle ingestion
- CDC & incremental loads
- Multi-tenant pipelines
- Unity Catalog / Lakehouse ingestion

◇ Interview One-Liner (Gold ✨)

"A metadata-driven ADF pipeline uses configuration tables to dynamically control data ingestion logic, allowing a single reusable pipeline to process multiple sources without code changes."

28. How to answer this, what are challenges you faced in azure data factory projects?

1️⃣ **Handling Dynamic & Large-Scale Ingestion**

Challenge:
We had to ingest data from **multiple sources** (SQL, ADLS, Dataverse) with different schemas and load patterns.

What I did:

- Implemented **metadata-driven pipelines**

- Used configuration tables for source, sink, and load type
- Reduced pipeline duplication

Result:
One generic pipeline handled **100+ tables**, easier maintenance and faster onboarding.

2️⃣ Incremental Load & Data Consistency

Challenge:
Ensuring **no data loss or duplication** during incremental loads.

What I did:

- Used **watermark columns** (LastModifiedDate)
- Stored watermark values in a control table
- Updated watermark only after successful loads

Result:
Reliable incremental loads with restart capability.

3️⃣ Performance Issues with Large Files

Challenge:
Slow copy performance while processing **large files / tables**.

What I did:

- Tuned **DIUs and parallelism**
- Used partitioning on source queries
- Switched to **Parquet format** for lake storage

Result:
Improved load performance significantly.

4️⃣ Error Handling & Monitoring

Challenge:
Pipeline failures were hard to debug initially.

What I did:

- Implemented **Try–Catch using If Condition**
- Logged errors into **Azure SQL / Log Analytics**
- Enabled alerts using Azure Monitor

Result:
Faster issue resolution and better operational visibility.

5️⃣ CI/CD & Environment Management

Challenge:
Managing deployments across **Dev, QA, Prod**.

What I did:

- Used **ARM templates**
- Parameterized linked services
- Integrated ADF with **Azure DevOps pipelines**

Result:
Smooth and controlled releases with minimal manual effort.

🏆 **One-Line Power Finish (Very Important)**

End with this:

"Overall, most challenges in ADF were around scalability, incremental processing, and operational reliability, which we addressed using metadata-driven design, proper monitoring, and performance tuning."

❌ **What NOT to Say**

- "ADF is slow"
- "I didn't face any challenges"
- Very tool-specific complaints without solutions

💡 **Want a Shorter Version?**

If the interviewer wants a **quick answer**, say:

"The main challenges I faced were building scalable pipelines, handling incremental loads, and monitoring failures. I solved these using metadata-driven pipelines, watermark-based incremental logic, and proper logging and alerting."

29. How to Answer "Performance Improvements in ADF"?

⭐ **Strong Sample Answer (Long / Real-World)**

"In Azure Data Factory projects, I worked on multiple performance improvements mainly around data movement, pipeline execution time, and scalability."

1️⃣ **Optimizing Copy Activity Performance**

Problem:
Large tables and files were taking too long to load.

Action:

- Tuned **DIUs and parallel copy**
- Enabled **partitioning** on source queries
- Increased **parallelism** for multiple tables

Result:
Load time reduced by **40–60%**.

2️⃣ Using Efficient File Formats

Problem:
Downstream processing on CSV was slow.

Action:

- Converted data to **Parquet** during ingestion
- Reduced file size and I/O

Result:
Improved read performance in Databricks and Synapse.

3️⃣ Metadata-Driven Parallel Processing

Problem:
Sequential pipelines caused delays.

Action:

- Implemented **metadata-driven pipelines**
- Used **ForEach with batch count**
- Enabled parallel execution for independent loads

Result:
Overall pipeline runtime reduced significantly.

4️⃣ Optimizing Linked Services & Integration Runtime

Problem:
Network latency and slow connections.

Action:

- Used **Azure IR** instead of Self-hosted where possible
- Chose correct **region alignment**
- Tuned IR compute size

Result:
Stable and faster data movement.

5️⃣ Avoiding Small File Problem

Problem:
Too many small files affected performance.

Action:

- Controlled file sizes using copy settings

- Merged files downstream in Databricks

Result:
Better lake performance and faster analytics.

🏆 **Strong Closing Line**

"Overall, performance improvements were achieved by optimizing copy activities, leveraging parallelism, using efficient file formats, and designing scalable metadata-driven pipelines."

⚡ **Short & Crisp Answer (1-Minute Version)**

"I improved ADF performance by tuning copy activity parallelism and DIUs, using partitioned reads, converting data to Parquet, and running pipelines in parallel using metadata-driven design. This reduced overall execution time by around 40–50%."

30. What is DIU?

DIU stands for **Data Integration Unit**.

In **Azure Data Factory**, **DIU is the unit of compute power used by the Copy Activity** to move and transform data.

🔹 **Simple Explanation (Interview-Friendly)**

DIU defines how much compute ADF uses for data movement.
Higher DIUs = more parallelism + higher throughput = faster copy.

🔹 **Where DIU Is Used**

- Only in **Copy Activity**
- Applies to:
    - Data extraction
    - Data loading
    - Light transformations (like column mapping, type conversion)

📌 DIU does **not** apply to:

- Mapping Data Flows
- Databricks activities
- Stored procedures

🔹 **DIU in Simple Terms**

Think of DIU like:

- 🚚 Number of trucks carrying data
  More trucks → faster delivery (but higher cost)

🔹 **DIU Range**

- Minimum: **2 DIUs**
- Maximum: **Up to 256 DIUs** (depends on IR and region)

🔹 **How DIU Impacts Performance**

DIU	Effect
Low DIU	Slower copy, cheaper
High DIU	Faster copy, higher cost
Too high	Diminishing returns

⚠️ Increasing DIU blindly doesn't always help—source/target limits matter.

🔹 **Real Interview Example Answer**

"DIU in Azure Data Factory represents the compute power used by the copy activity. By tuning DIUs along with partitioning and parallel copy, we can significantly improve data movement performance while balancing cost."

🔹 **DIU vs Parallel Copy (Very Important)**

DIU	Parallel Copy
Controls compute power	Controls concurrency
Vertical scaling	Horizontal scaling
Cost-based	Throughput-based

✅ Best practice: **Tune both together**

🔹 **Best Practices (Say This in Interview)**

- Start with default DIU
- Measure throughput
- Increase gradually
- Combine with partitioning
- Avoid over-provisioning

31. Did you use mapping dataflows?

"Very selectively. In most projects, we preferred Copy Activity + Databricks. Mapping Data Flows were used only where no-code transformations were sufficient and cost justified."

32. When would you use Mapping Data Flows in ADF?

- Simple transformations
- No complex business logic
- Low to medium data volume
- Teams avoiding Spark code

33. When would you NOT use Mapping Data Flows? ⭐ (Very Important)

- Complex transformations
- Heavy joins & aggregations
- Large volumes
- Cost-sensitive workloads
- When Databricks is already available

34. Why are Mapping Data Flows expensive?

- Spark cluster spins up per run
- Charged per vCore-hour
- Cold start latency
- Debug mode costs

35. How do you handle performance tuning in Data Flows?

- Partitioning
- Broadcast joins
- Cache sink
- Proper compute sizing

36. Why are Mapping Data Flows not popular in real projects?

- Cost
- Debug difficulty
- Less flexibility than Spark
- Limited transformations
- Teams already use Databricks

37. What happens behind the scenes when a Data Flow runs?

- ADF provisions an ephemeral (short lived) Spark cluster
- Executes transformations
- Shuts down after completion

38. Best answer if asked view on mapping dataflows?

"In real projects, Mapping Data Flows are used sparingly. Most enterprises prefer Copy Activity for ingestion and Databricks for transformation. Mapping Data Flows fit well only for simple, no-code transformation scenarios."

39. What not to say on mapping dataflows?

- "Mapping Data Flows are useless"
- "I never learned them"
- "ADF transformations are bad"

40. What are differences between copy activity and mapping dataflows?

🔹 **Copy Activity vs Mapping Data Flows (ADF)**

Aspect	**Copy Activity**	**Mapping Data Flows**
Purpose	Data **movement**	Data **transformation**
Compute	DIU (Data Integration Units)	Spark clusters (vCores)
Cost	Low	High
Performance	Very fast for ingestion	Slower due to cluster startup
Transformations	Basic (mapping, type cast, column select)	Complex (joins, agg, derive)
Scalability	High for ingestion	High but cost-heavy
Coding	No-code	No-code (Spark underneath)
Startup latency	Almost none	Cold start (1–3 mins)
Debugging	Easy	Comparatively difficult
Real-world usage	**Very high**	**Limited / selective**

🔹 **When to Use Copy Activity**

✓ Real-world, enterprise scenarios:

- Source → Data Lake ingestion
- Large volume data movement
- Metadata-driven pipelines
- Incremental loads
- Cost-sensitive pipelines

Most production pipelines are Copy-Activity-heavy

🔹 **When to Use Mapping Data Flows**

✓ Selective scenarios:

- Simple joins & aggregations
- Business users / no Spark skills
- Small to medium datasets
- Prototyping transformations

🧠 What Interviewers *Really* Want to Hear?

Use this line:

"In real projects, we primarily use Copy Activity for ingestion due to its performance and cost efficiency. Mapping Data Flows are used selectively for simple transformations, while complex logic is handled in Databricks."

One-Line Answer (Perfect for Fast Rounds)

"Copy Activity is optimized for fast, cost-effective data movement, while Mapping Data Flows are Spark-based and used only when no-code transformations are required."

41. "What data sizes did you handle?"

"Data sizes varied by source. Some tables were a few GBs, while fact tables ranged from tens of GBs to hundreds of GBs per load."

42. How do you handle large data in ADF?

"For large datasets, we focus on partitioned reads, parallel copy, DIU tuning, and efficient file formats like Parquet. We also avoid heavy transformations in ADF and offload them to Databricks."

43. What changes when data size increases?

"As data size grows, we adjust partitioning, increase parallelism, tune DIUs, use incremental loads instead of full loads, and ensure downstream systems can scale."

44. Why Parquet over CSV?

"We prefer Parquet over CSV because Parquet is a columnar storage format. It supports compression, schema enforcement, and predicate pushdown, which makes downstream analytics in tools like Databricks, Synapse, and Spark much faster and more cost-efficient compared to CSV."

🔹 **Key Differences (Say These Keywords)**

Aspect	CSV	Parquet
Storage type	Row-based	**Columnar**
Compression	Poor / manual	**Built-in**
Schema	No schema	**Schema-aware**
Query performance	Slow	**Fast**
Predicate pushdown	❌	✅
Storage cost	Higher	**Lower**
Big data suitability	❌	✅

45. Projects to be mentioned in resume.

✅ **PROJECT 1: Enterprise Data Lake Ingestion (ADF-Centric)**

Project Title

Metadata-Driven Data Ingestion Framework using Azure Data Factory

Domain

Retail / E-commerce Analytics

Tech Stack

Azure Data Factory, ADLS Gen2, Azure SQL DB, Azure Monitor, Databricks

Data Scope

- **No. of tables:** ~120 tables
- **Data size:** 5–80 GB per table (daily loads)
- **Load type:** Full (initial) + Incremental (daily)

Resume Description

- Designed and implemented a **metadata-driven ingestion framework** in Azure Data Factory to ingest data from Azure SQL Database into ADLS Gen2.
- Built **generic ADF pipelines** using Lookup and ForEach activities to process 100+ tables dynamically.
- Implemented **incremental loading using watermark columns**, with control tables to track last processed values.
- Optimized performance using **partitioned reads, parallel copy, and DIU tuning**, reducing pipeline runtime by ~40%.
- Stored ingested data in **Parquet format** to improve downstream analytics performance in Databricks.

- Implemented **error handling, logging, and retry mechanisms**, with monitoring via Azure Monitor.

✅ **PROJECT 2: Finance / ERP Data Integration Platform**

Project Title

Scalable Financial Data Integration using Azure Data Factory

Domain

Finance / ERP Reporting

Tech Stack

Azure Data Factory, ADLS Gen2, Azure SQL DB, Azure Key Vault, Azure DevOps

Data Scope

- **No. of tables:** ~60 tables
- **Data size:** 1–25 GB per table
- **Load type:** Full refresh + Incremental (hybrid)

Resume Description

- Developed ADF pipelines to ingest financial and transactional data from on-prem and cloud sources into a centralized data lake.
- Implemented **hybrid loading strategy** with full refresh for reference tables and incremental loads for transaction tables.
- Parameterized datasets and linked services to support **multi-environment deployments (Dev/QA/Prod)**.
- Integrated **Azure Key Vault** for secure credential management.
- Implemented **CI/CD pipelines using Azure DevOps** for automated ADF deployments.
- Improved data availability and reduced manual intervention through **robust monitoring and alerting mechanisms**.

✅ **PROJECT 3: Analytical Platform (ADF + Databricks)**

Project Title

End-to-End Analytics Platform using ADF and Databricks

Domain

Sales & Customer Analytics

Tech Stack

Azure Data Factory, ADLS Gen2, Databricks (Spark), Delta Lake

Data Scope

- **No. of tables:** ~40 tables
- **Data size:** 10–150 GB (fact tables)
- **Load type:** Incremental + Historical Backfill

Resume Description

- Used Azure Data Factory as an **orchestration layer** to trigger and manage Databricks-based transformations.
- Ingested raw data into ADLS using ADF Copy Activity and converted it to **Parquet/Delta format** for analytics.
- Implemented **incremental ingestion with backfill support** for historical data reprocessing.
- Addressed **small file issues** by controlling file sizes and consolidating data in Databricks.
- Ensured pipeline **idempotency and restartability** to handle partial failures without data duplication.
- Collaborated with analytics teams to deliver curated datasets for reporting and dashboards.