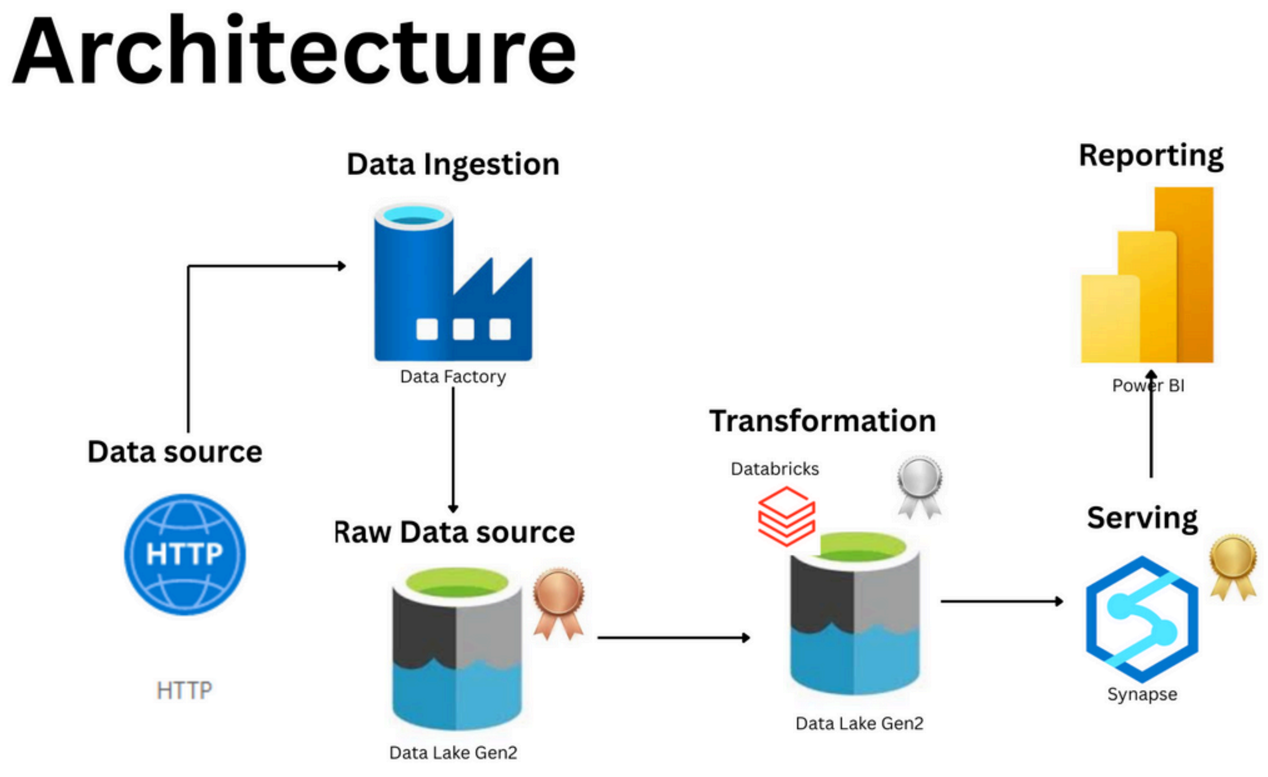
Definitions of Terms Used in the Project(end to end Data pipeline)



Orchestration in ELT :

Orchestration means managing and automating the steps in the ELT process.

Data Lake Architecture / Data Engineering / Big Data Pipeline Design

Medallion Layer	Purpose	Type of Data	Related Term
Bronze Layer	Raw data ingestion	Raw (unprocessed)	**Raw Layer**
Silver Layer	Data cleaning, filtering, joining	Transformed data	**Transformation Layer** or **Cleansed Layer**
Gold Layer	Aggregated, business-ready data	Curated (analytics)	**Serving Layer** or **Presentation Layer**

Resource group

A container that holds related resources for an Azure solution.

tags

Apply tags to your Azure resources to logically organize them by categories.

Storage Account

Create a storage account to store up to 500TB of data in the cloud.

GRS (Geo-Redundant Storage)

GRS replicates your data to a secondary region (hundreds of miles away from the primary region) to ensure durability and high availability, even in case of a regional disaster.

LRS (Locally Redundant Storage)

LRS (Locally Redundant Storage) is a redundancy option in Azure that stores multiple copies of your data within a single data center (in one region).

Hierarchical namespace

Hierarchical namespace in Azure Data Lake Storage Gen2 organizes data into directories and files, similar to a traditional file system. This structure improves performance for big data analytics and supports fine-grained security using Access Control Lists (ACLs).

📦 **Azure Blob Storage** - Object storage:

Blob = Binary Large Object

Standard Azure Blob Storage endpoint.

Used for general-purpose access (upload, download, etc.)

Good for unstructured data:

- Images
- Videos
- Backups
- Log files
- CSV/Parquet files (for uploading/access)

Storage Account

└──── Container

  └──── Blob (file)

🌊 **Azure Data Lake Storage Gen2** - Big data analytics storage built on top of Blob Storage:

DFS = Distributed File System

Endpoint for **Azure Data Lake Storage Gen2**

Enables:

- Directory/folder structure
- Fine-grained file access (ACLs)
- Optimized for big data analytics (Synapse, Databricks, etc.)

Required when using storage as a **data lake** (with OPENROWSET, Synapse, etc.)

Storage Account

└──── File System (container)

```
└──── Directory (folder)
    └──── Sub-directory (optional)
        └──── File
```

DATA FACTORY (ADF)

🔹 **Home** – Provides a quick overview and shortcuts to create pipelines, datasets, and other resources.
🔹 **Author** – Used to create and edit pipelines, data flows, datasets, and triggers visually.
→ **ADF Data Flow (Mapping Data Flow)** – A visual data transformation tool in Azure Data Factory that allows you to design and execute ETL (Extract, Transform, Load) logic without writing code.
🔹 **Monitor** – Lets you track and manage pipeline runs, debug logs, and activity statuses.
🔹 **Manage** – Handles linked services, integration runtimes, triggers, and other configuration settings.
🔹 **Learning Center** – Offers tutorials, documentation, and quick-start guides to help you learn and explore ADF features.

STORAGE ACCOUNT

🔹 **Containers** – Used in Blob Storage to store unstructured data like documents, images, and videos in the form of blobs.
🔹 **File Shares** – Part of Azure Files, they provide shared storage accessible via SMB protocol, like a cloud file server.
🔹 **Queues** – Used in Azure Queue Storage to store and manage messages for asynchronous communication between applications.
🔹 **Tables** – Part of Azure Table Storage, used to store structured NoSQL data with key-value pairs, ideal for fast lookups.

ACTIVITIES in ADF

Activities are the building blocks of a pipeline,they perform specific tasks like data movement, data transformation, or control flow.

Copy Activity in ADF

The **Copy Activity** is used to move or copy data from a source (like a database, file, or API) to a destination (like Azure Data Lake, Blob Storage, or SQL Database).

Source – The location from where data is read or extracted in the Copy Activity.

Sink – The destination where data is written or loaded after copying.

LINKED SERVICE

A **Linked Service** is like a connection string that defines how ADF connects to external data sources or compute services.

Static Pipeline –

- Works with hardcoded values like fixed file names, table names, or paths.
- Not reusable — needs editing if the source/sink changes.

Dynamic Pipeline –

- Uses parameters, variables, and expressions to make the pipeline flexible and reusable.
- Adapts to different inputs without changing the pipeline structure.

URL: Base and Relative

- **Base URL** is the main part of a web address (e.g., https://example.com/)
- **Relative URL** is the path added to the base to reach a specific resource (e.g., /images/logo.png)

DATASET in Azure Data Factory:

Purpose

A Dataset in ADF defines the structure and location of the data you want to use—whether for reading (source) or writing (sink).

MAPPING in ADF (Azure Data Factory)

Mapping defines how columns/fields from the source are matched or transformed into the destination (sink) schema during data copy or transformation.

IMPORT SCHEMA in ADF

Import Schema automatically reads the structure (columns and data types) from the source or sink and loads it into the dataset.

PUBLISH ALL

In author in Azure Data Factory (ADF), the "Publish All" button is used to save and apply all changes you've made in the Author tab to the live data factory environment.

ITERATIONS AND CONDITIONS

In Azure Data Factory (ADF), **iterations and conditions** are used in pipelines to control the flow of execution, enabling dynamic, automated, and logic-driven data workflows.

FOREACH ACTIVITY with Iterations and Conditions in ADF

Use **ForEach** to loop through a list of files and dynamically copy data based on:

- Different **relative URLs**
- Target **folders**
- Target **file names**

ITEMS PROPERTY

The **Items** property in a ForEach activity specifies the **array or list** of elements that the loop will iterate over.

LOOKUP Activity in Azure Data Factory (ADF)

The **Lookup** activity is used to retrieve data from a source and make it available for use in the pipeline

Visually connect activities using the canvas

Node	Explanation
On Skip	Executes when the current activity is skipped (used in conditional branches).
Success	Runs the next activity **only if the previous one succeeds**.
Failure	Triggers the next activity **only if the previous one fails**.
Completion	Runs the next activity **regardless of success or failure**.

Managed Resource Group in Azure Databricks

- It is an **Azure resource group automatically created and managed by Databricks**.
- It holds **dependent resources** like **virtual networks, storage**, etc., used by the workspace.
- **All transformations run on the cluster**, but we **don't manage the cluster ourselves**—Databricks takes care of it, including managing all the **Virtual Machines (VMs)** and the **Virtual Network (VNet)**.

In Azure Databricks: Key Components

- **Compute**:
 Refers to the clusters of virtual machines used to run data processing and analytics workloads.
- **Catalog**:
 A centralized data governance layer in Unity Catalog that organizes data into schemas and tables, ensuring secure and discoverable access across workspaces.
- **Workspace**:
 The environment where users collaborate through notebooks, dashboards, libraries, and jobs; it's the UI for managing code, data, and results.
- **Workflow**:
 A set of scheduled or triggered jobs and tasks in Databricks that automate data pipelines, notebook executions, and other processes in a defined sequence.

Purpose of Databricks

The primary purpose of Databricks is to simplify and accelerate big data and AI workflows by providing a unified analytics platform.

In Azure Databricks Compute

- **Single-node cluster**:
  A cluster with only one virtual machine that runs both the driver and worker processes. Ideal for development, testing, or small workloads.
- **Multi-node cluster**:
  A cluster with one driver node and multiple worker nodes that distribute the workload for parallel processing. Used for production-scale and large data processing tasks to improve performance and scalability.

Understanding the data loading Path:

.load("abfss://bronze@dpstoragedatalake.dfs.core.windows.net/AdventureWorks_Calendar")

abfss://

- Stands for **Azure Blob File System Secure**
- Used to connect **securely** to **ADLS Gen2** with **OAuth authentication**

bronze@dpstoragedatalake.dfs.core.windows.net

- bronze: The **container** (similar to a folder in storage)
- dpstoragedatalake.dfs.core.windows.net: The **fully qualified domain name** of your **Azure Data Lake Storage** account

/AdventureWorks_Calendar

- The **file or folder path** inside the bronze container where your CSV file(s) are stored

withColumn() Function

The withColumn() function is a powerful way to **add a new column** or **update an existing column** in a Spark DataFrame.

📌 **Syntax:**

```
df = df.withColumn("new_column_name", expression_or_transformation)
```

Month() Function

In PySpark, the month() function is used to **extract the month** (1 to 12) from a **date or timestamp** column.

📌 **Syntax:**

```
df = df.withColumn("month_column", month(df["your_date_column"]))
```

Year() Function

The year() function extracts the **year** (e.g., 2023) from a **date or timestamp** column.

📌 **Syntax:**

```
df = df.withColumn("year_column", year(df["your_date_column"]))
```

write

- write is an **action in PySpark** used to **save or export a DataFrame** to external storage (like a file system, Azure Data Lake, S3, etc.).
- It supports **different file formats** such as **CSV, Parquet, JSON**, etc.

format("parquet")

- Specifies the **file format** in which you want to save the data.
- "parquet" is a **columnar storage format** widely used in big data environments because:
  - It's **highly efficient** in terms of **space and speed**.
  - It **preserves schema**.
  - It **supports compression**.
  - It's ideal for **analytical queries** (especially in Spark, Hive, etc.).
- **Best choice** for **structured data** and **performance-critical applications**.

mode() in pyspark

- Specifies the **behavior** when the **output file/directory already exists**.
- There are **4 write modes** in PySpark:

Mode	Description
"overwrite"	Overwrites the existing data at the location.
"append"	Adds new data to the existing data (does not overwrite).
"ignore"	If data already exists, it skips writing without throwing an error.
"error" or "errorifexists"	Default mode. Throws an error if data already exists.

split() Function

- The split() function in PySpark is used to **split a string column** into an **array of substrings** based on a given delimiter.

📌 **Syntax:**

split(column, pattern)

column: The column to split (e.g., col("FullName"))

pattern: The delimiter or regex pattern to split by (e.g., " ", "-", ",")

concat(...)

- This combines **multiple columns** as-is into a **single string column**.
- Unlike concat_ws, concat does **not insert separators automatically**.
- That's why lit(" ") is used between each column to **add spaces manually**.

col("Prefix"), lit(" "), col("FirstName"), lit(" "), col("LastName")

Combines the values in order:

Prefix + " " + FirstName + " " + LastName

Example:
"Mr." + " " + "John" + " " + "Doe" → **"Mr. John Doe"**

.display()

- This is **specific to Databricks notebooks**.
- It renders the resulting **DataFrame as a table** in the UI.
- It’s **more interactive** than .show() which just prints rows in the console

concat_ws(" ", ...)

- concat_ws means **concatenate with separator**.
- Here, the separator is " " (a space), so it joins the given columns with spaces.

to_timestamp('StockDate')

- This function converts the 'StockDate' column from a **string** (or other format) to a **timestamp type**.
- A timestamp includes both **date and time** information (e.g., 2025-05-31 14:23:00).
- Internally, this allows you to perform **date-time functions** like:
  - **Filtering**
  - **Difference calculation**
  - **Sorting**

regexp_replace('OrderNumber', 'S', 'T')

- regexp_replace() is a **PySpark SQL function** used to **replace all occurrences** of a regex pattern in a string column.

In this case:

- It looks for all **capital 'S'** characters in the 'OrderNumber' string.
- **Replaces** each 'S' with 'T'.

groupBy('OrderDate')

- This **groups** the df_sale DataFrame by each **unique value** in the OrderDate column.
- All rows that have the **same OrderDate** are grouped together for aggregation.

.agg(count('OrderNumber').alias('count'))

- After grouping, we perform an **aggregation** using .agg():
    - count('OrderNumber'): Counts how many **orders** (i.e., **non-null values**) exist for each OrderDate.
    - .alias('count'): Renames the resulting column to "count" for readability.

What is Azure Synapse?

Azure Synapse Analytics is an integrated analytics service combining:

- **Data integration** (like pipelines in Data Factory),
- **Big data** (like Spark),
- **Data warehousing** (with dedicated or serverless SQL pools).

Synapse Studio Sections:

🔹 **Home**

Dashboard overview of the workspace with quick links to create SQL scripts, pipelines, notebooks, etc.

🔹 **Data**

Explore and manage data. Access linked Azure Data Lake, SQL pools, create tables, views, or databases.

🔹 **Develop**

Write and run SQL scripts, Spark notebooks, data flows, and Kusto queries.

🔹 **Integrate**

Create and manage data pipelines. Orchestrate workflows, move data, and schedule tasks.

🔹 **Monitor**

View status/history of all activities (pipeline runs, triggers, SQL jobs, Spark jobs) for debugging.

🔹 **Manage**

Configure Synapse workspace. Manage linked services, SQL pools, credentials, integration runtimes, security.

What is a System-Assigned Managed Identity in Synapse?

A **System-Assigned Managed Identity** is an **automatically created identity** tied to your Synapse workspace that lets Synapse **securely access other Azure services** (like Azure Data Lake, Key Vault, etc.) without storing credentials.

Serverless SQL Pool vs Dedicated SQL Pool

Feature	**Serverless SQL Pool**	**Dedicated SQL Pool**
Provisioning	No provisioning required (auto-managed by Azure)	Must be provisioned manually with defined compute resources
Pricing	**Pay-per-query** ($ per TB of data processed)	**Pay-per-hour** (based on performance level DWU)
Use Case	Best for **ad-hoc queries** on files in Data Lake	Best for **large-scale data warehousing & structured data**
Performance	Shared resources (slower for big workloads)	High performance, **dedicated compute**
Data Storage	Reads data from **external files** (Parquet, CSV, etc.)	Stores data in **Synapse tables**
Management	Fully managed; no scaling or maintenance	Requires **scaling and performance management**
Startup Time	Instant	May take time to resume or scale

Data Lake

A **Data Lake** is a **centralized storage repository** that holds **large volumes of raw data** in its **native format** (structured, semi-structured, or unstructured), typically stored in files like CSV, JSON, Parquet, images, videos, etc.

Lakehouse

A **Lakehouse** is a modern data architecture that **combines the features of a Data Lake and a Data Warehouse** into a single platform.

A **Lakehouse architecture** provides an **abstraction layer** that **combines the storage of a Data Lake** with the **structured processing and transactional features of a Data Warehouse**.

Why Data Lake?

Because it reduces cost — cheaper than storing data in databases.

OPENROWSET() in Azure Synapse

OPENROWSET() is a **T-SQL function** in Azure Synapse Analytics that lets you **query external data files** (like CSV, Parquet, or JSON) stored in **Azure Data Lake** or **Blob Storage without loading them into a table**.

Managed Table

A **Managed Table** is a table **stored within Synapse's internal storage** (either dedicated SQL pool or serverless). When you create it, **Synapse manages the data and metadata**.

External Table

An **External Table** is a table **points to data stored outside Synapse**, typically in **Azure Data Lake Storage (ADLS Gen2)** or **Blob Storage**.

CETAS(Create External Table As Select)

CETAS allows us to **create an external table** and **push data** at the same time using a SELECT query.

📌 Syntax

CREATE EXTERNAL TABLE gold.extsales

WITH (

```
    LOCATION = 'extsales',

    DATA_SOURCE = source_gold,

    FILE_FORMAT = format_parquet

)

AS

SELECT * FROM silver_view;
```

DATABRICKS

Databricks is a cloud-based data platform built on Apache Spark that integrates data engineering, machine learning, and analytics .It allows teams to collaborate in a unified workspace to process large-scale data efficiently.