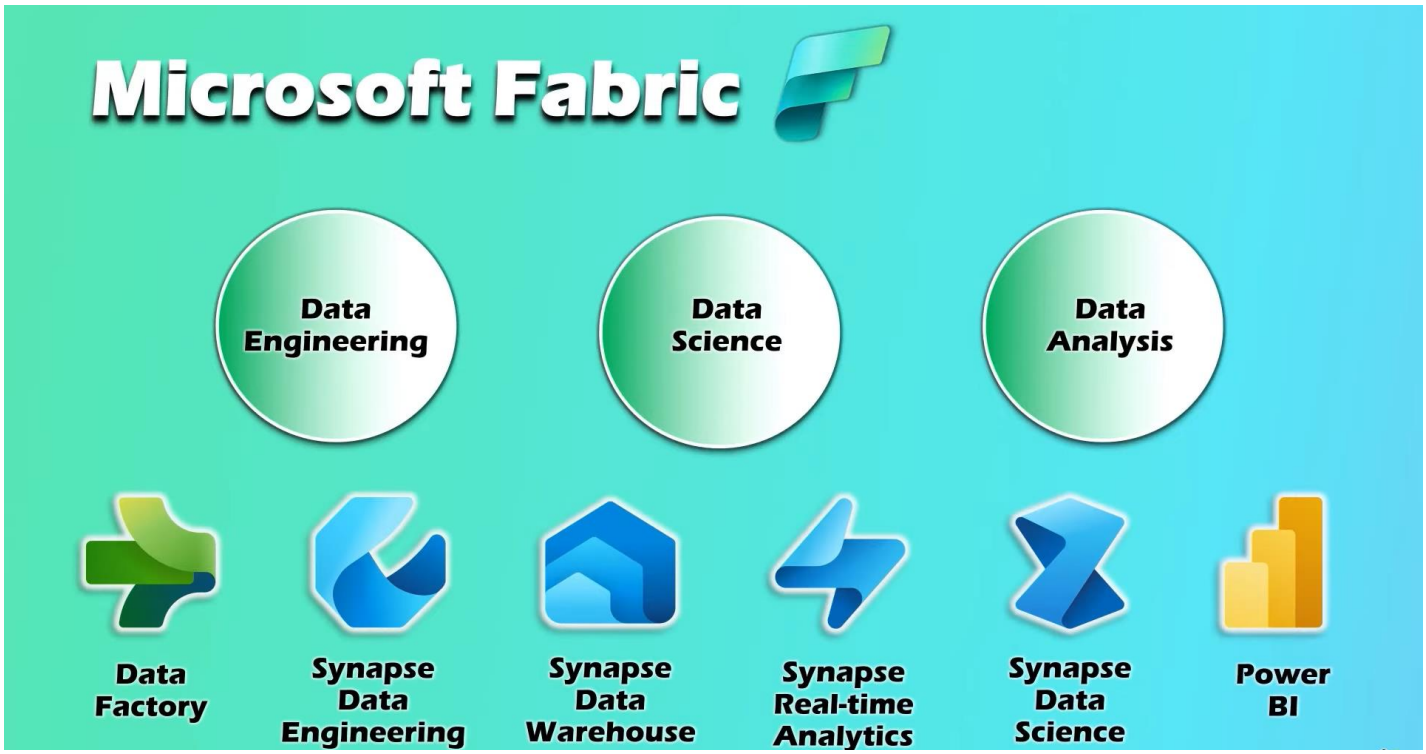
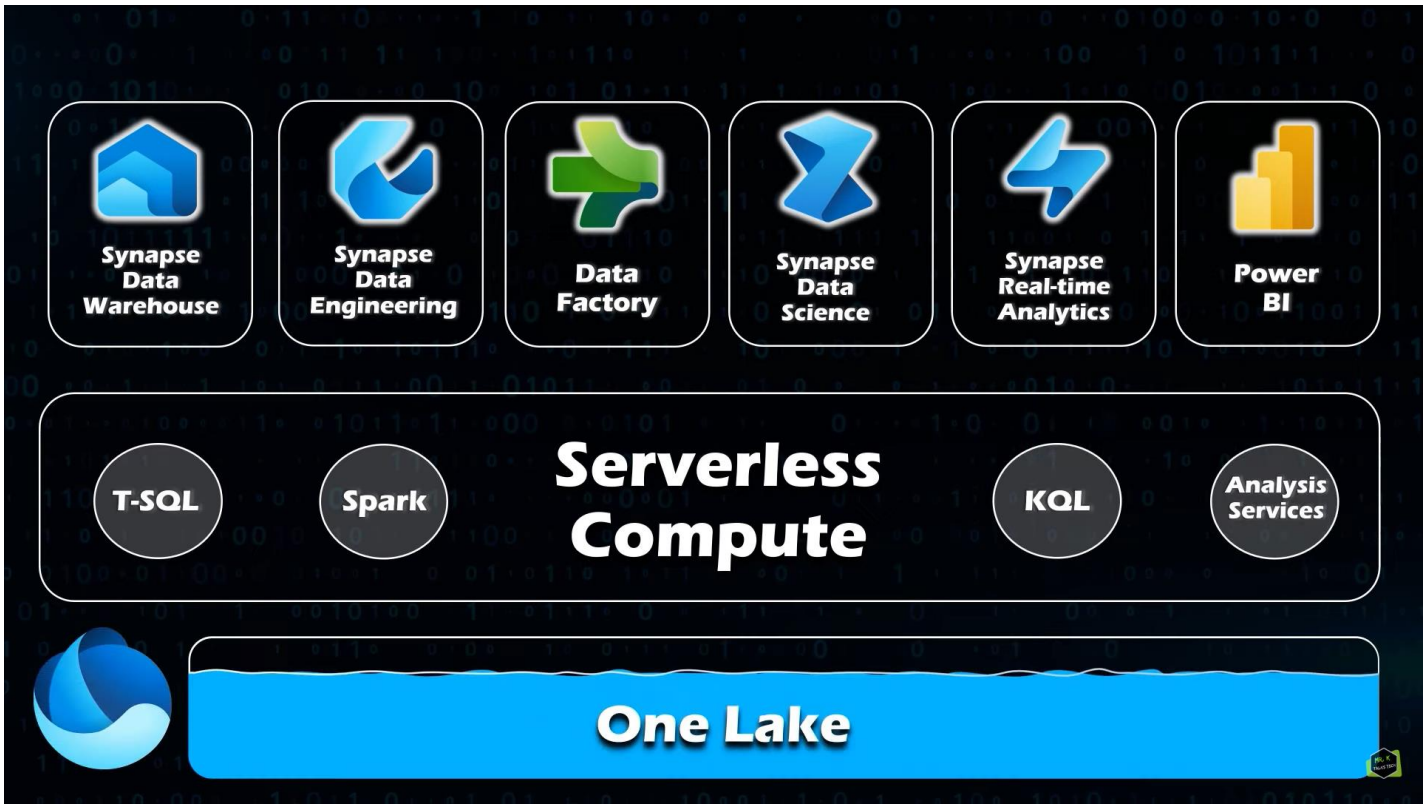
Microsoft Fabric Tutorials

Part 1: What is Microsoft Fabric





Microsoft Fabric

- It is an all-in-one unified analytics platform

Data Engineering

Data Science

Data Analysis

Microsoft Fabric

Data Engineering

Data Science

Data Analysis

Data Factory

Synapse Data Engineering

Synapse Data Warehouse

Synapse Real-time Analytics

Synapse Data Science

Power BI

Data Factory

- Pipelines
- Data Flows

Synapse Data Engineering

- Notebooks
- Spark Job Definition
- Lakehouse
- Pipeline

Synapse Data Warehouse

- Warehouse
- Pipeline

Synapse Data Science

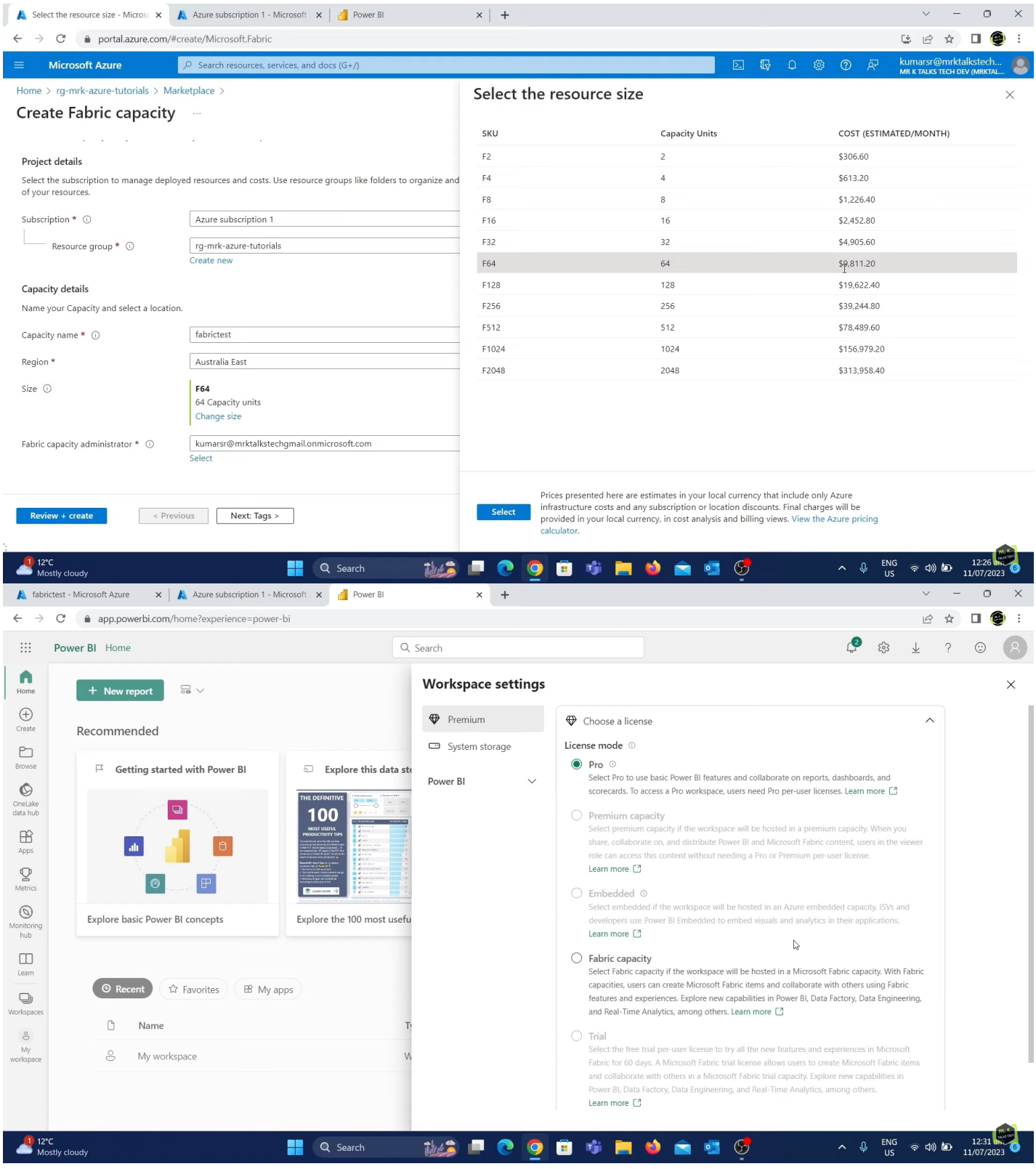
- Model
- Experiment
- Notebook

Synapse Real Time Analytics

- KQL DB
- Eventstream

Part 2:
Create and Enable Microsoft Fabric using Azure Portal

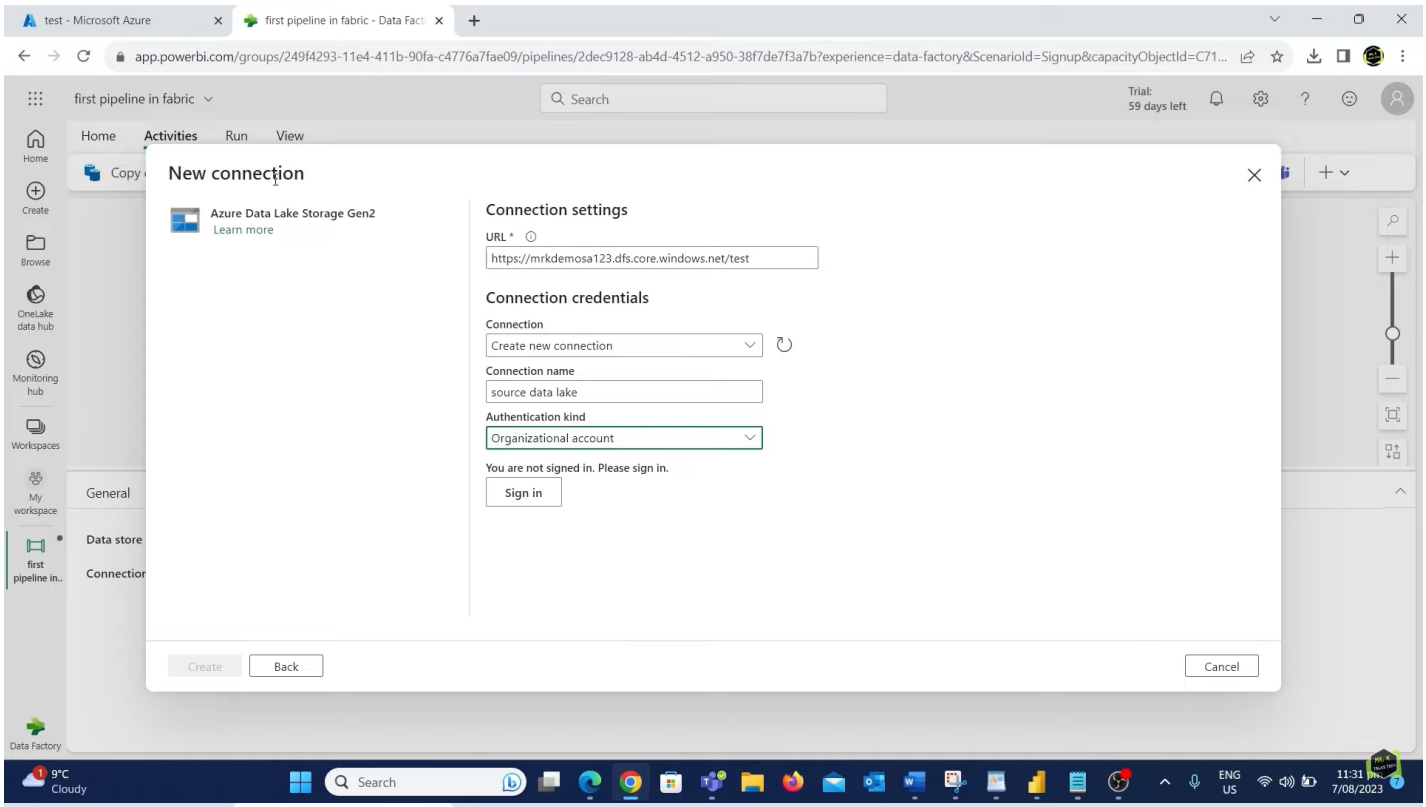
Goto azure account--->goto resource--->search Microsoft Fabric--->azure subscriptions--->resource providers--->search fabric--->register--->click ok

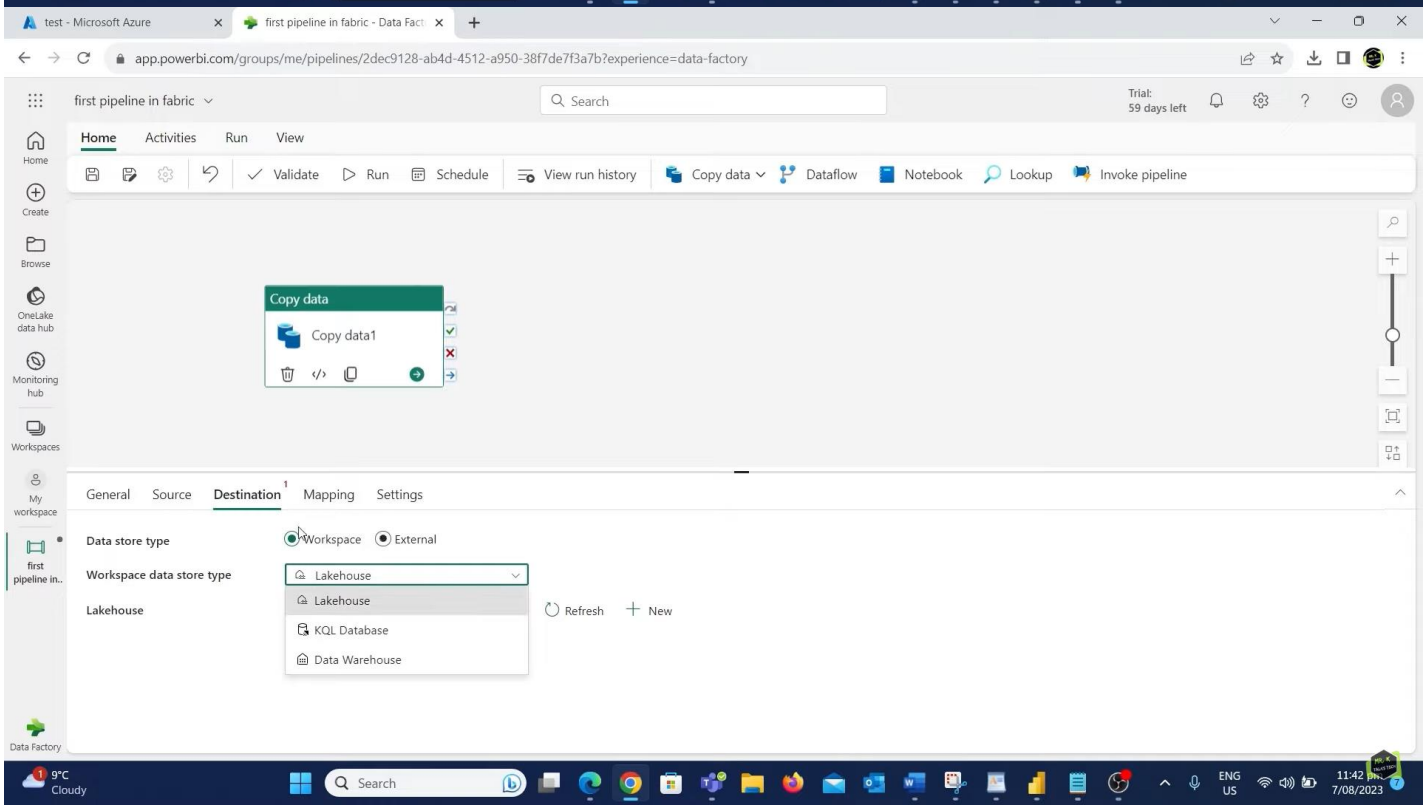


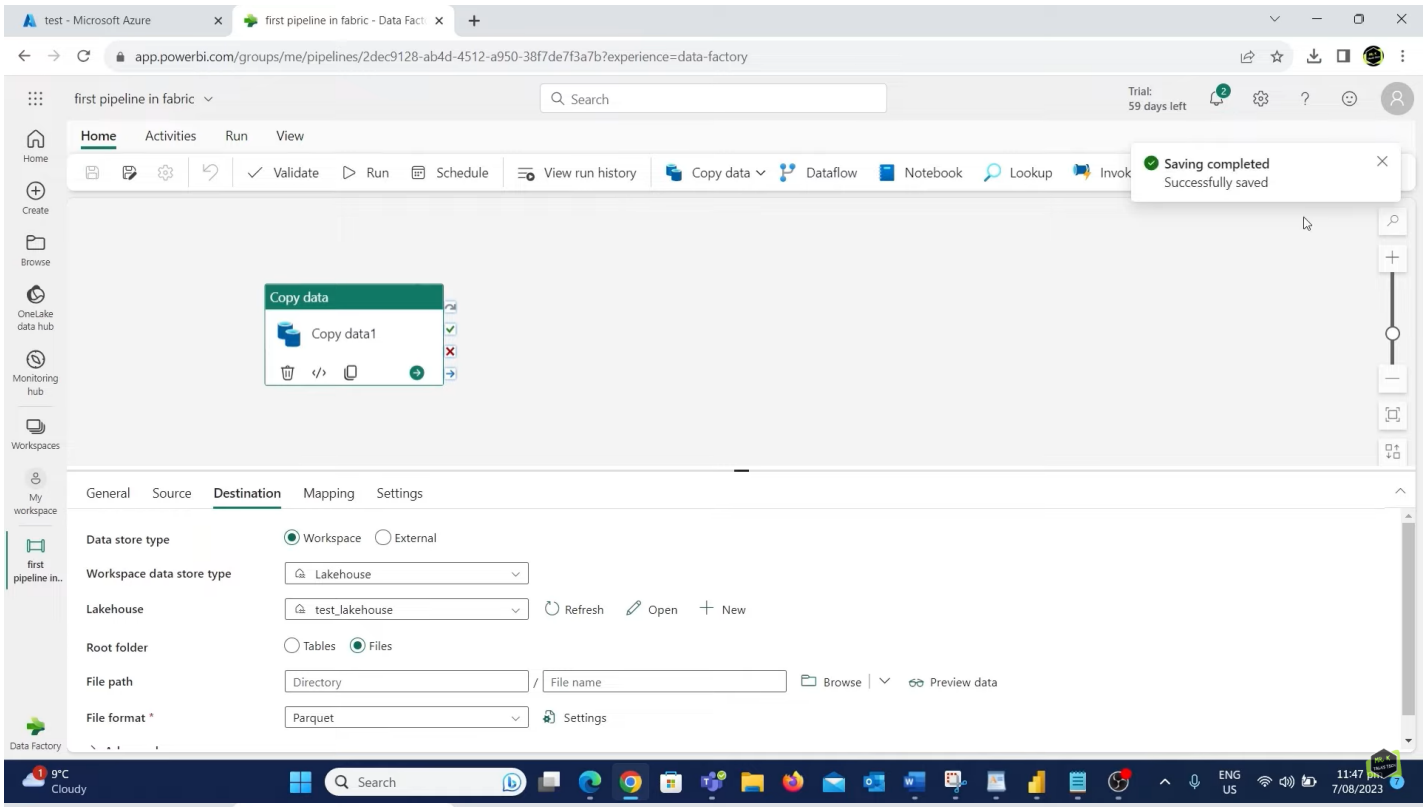
Part 3:
Create your first Data Ingestion Pipeline in Microsoft Fabric.

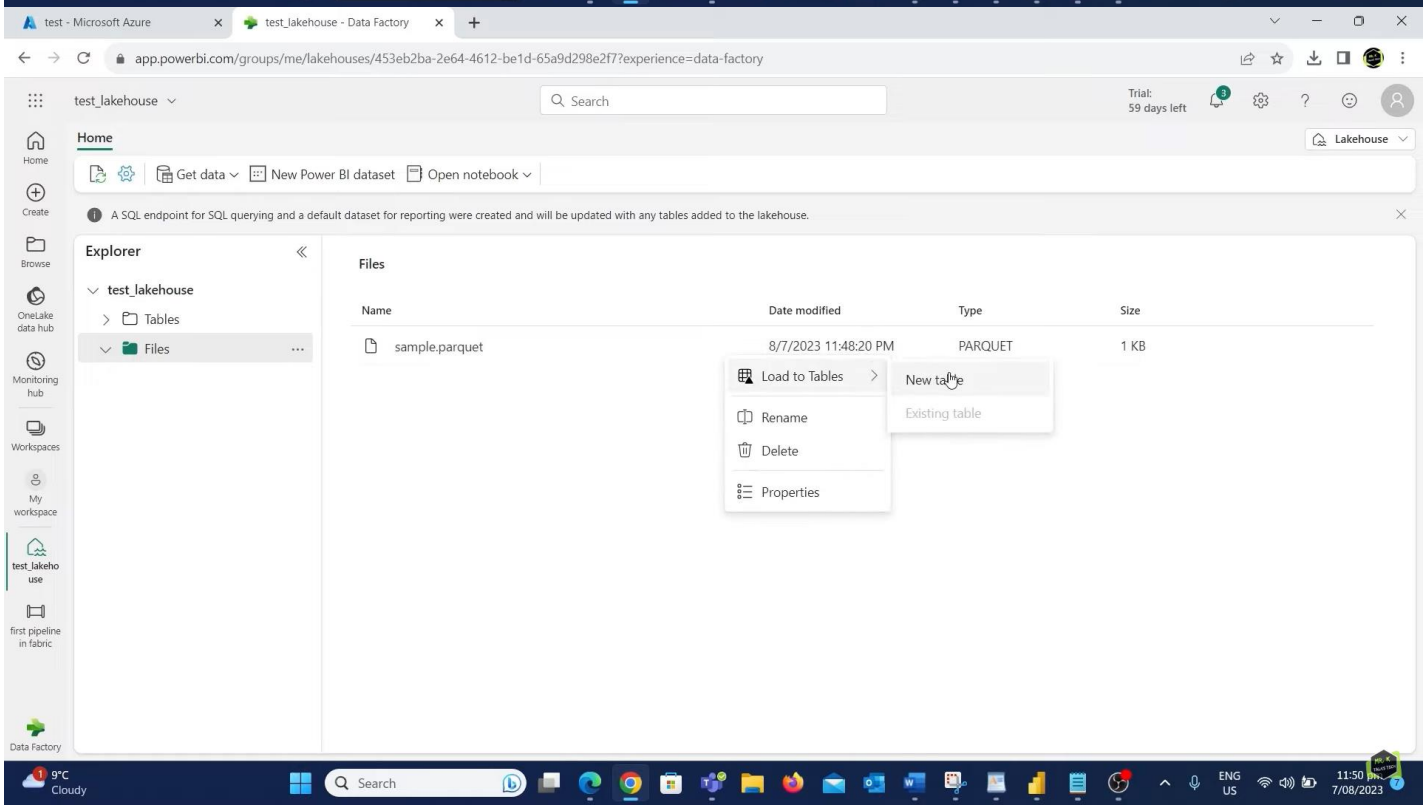
Goto azure portal--->azure ADLS Gen2--->container--->properties--->copy URL

New pipeline--->copy activity--->source--->connection--->ADLS Gen2--->Paste URL


test - Microsoft Azure
first pipeline in fabric - Data Fact
app.powerbi.com/groups/249f4293-11e4-411b-90fa-c4776a7fae09/pipelines/2dec9128-ab4d-4512-a950-38f7de7f3a7b?experience=data-factory&ScenarioId=Signup&capacityObjectId=C71...
first pipeline in fabric
Search
Trial: 59 days left
Home
Activities
Run
View
New connection
Azure Data Lake Storage Gen2
Learn more
Connection settings
URL *
https://mrkdemosa123.dfs.core.windows.net/test
Connection credentials
Connection
Create new connection
Connection name
source data lake
Authentication kind
Organizational account
You are not signed in. Please sign in.
Sign in
Create
Back
Cancel
General
Data store
Connection
Data Factory
9°C Cloudy
Search
ENG US
11:31
7/08/2023


test - Microsoft Azure
first pipeline in fabric - Data Fact
app.powerbi.com/groups/me/pipelines/2dec9128-ab4d-4512-a950-38f7de7f3a7b?experience=data-factory
first pipeline in fabric
Search
Trial: 59 days left
Home
Activities
Run
View
Validate
Run
Schedule
View run history
Copy data
Dataflow
Notebook
Lookup
Invoke pipeline
Copy data
Copy data1
General
Source
Destination
Mapping
Settings
Data store type
Workspace
External
Workspace data store type
Lakehouse
Lakehouse
Lakehouse
KQL Database
Data Warehouse
Refresh
New
Data Factory
9°C Cloudy
Search
ENG US
11:42
7/08/2023





Delta Lake:

Delta Lake = Data Warehouse + Data Lake

Data Warehouse

- Only Structured Data
- Schema-on-Write

- Supports ACID Transaction
- Does not corrupt the system

Data Lake

- Structured/Semi-structured/Unstructured
- Schema-on-Read
- Minimal support to ACID Transaction
- Leaves system in corrupted state

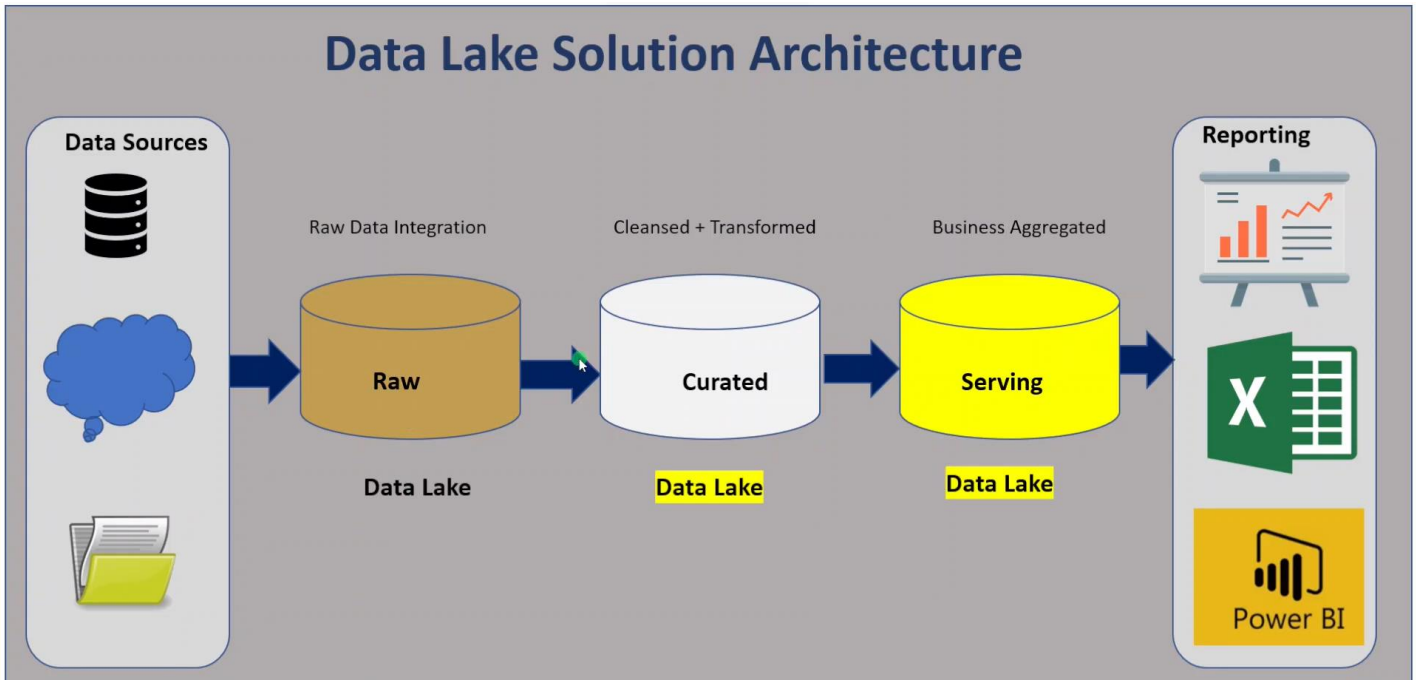
Delta Lake

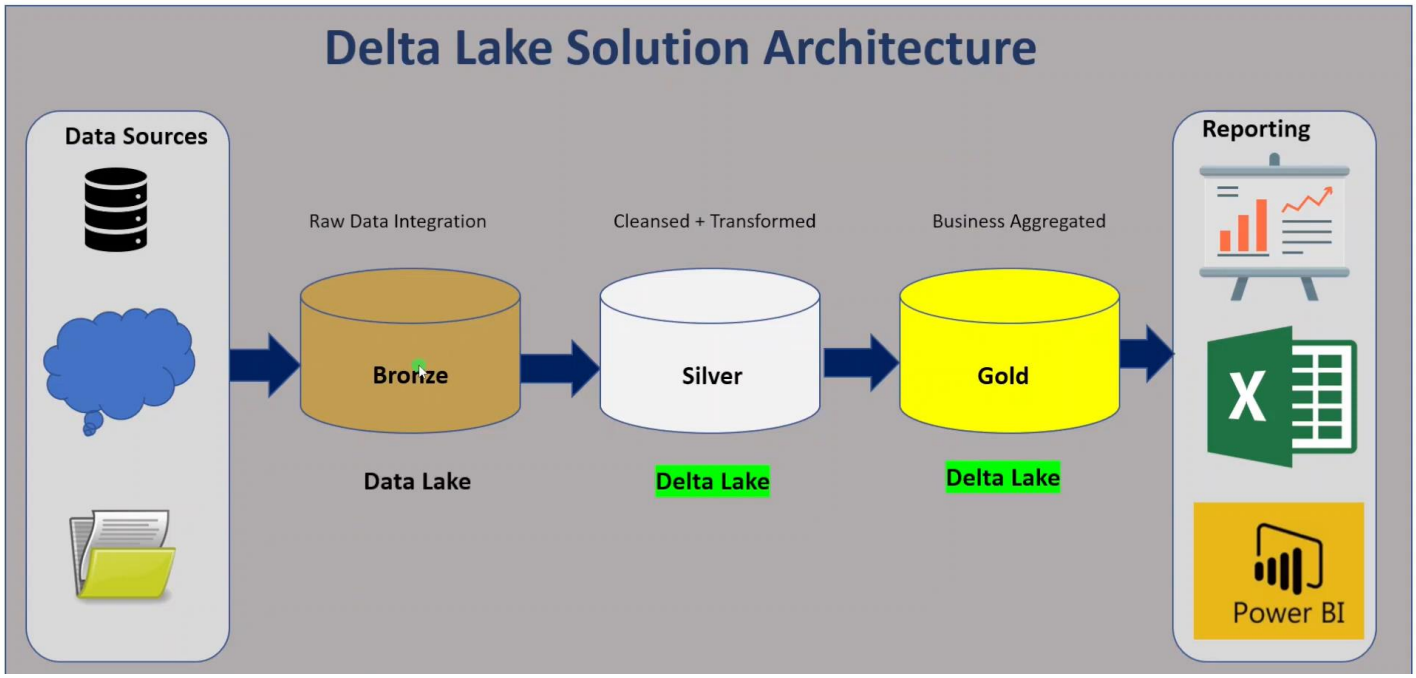
- Structured/Semi-structured/Unstructured/Streaming/Batch Processing
- Schema-on-Read
- Supports ACID Transaction
- Does not corrupt the system
- Open-source storage layer on top of data lake mainly used to support
- ACID Transactions.
- Ensures data reliability
- Simplify Lambda architecture for Batch+ Streaming Combination.
- DML operations (Insert, Update, Delete, Merge)

Json Transaction Log File

Parquet Checkpoint File

CRC — Cyclic Redundant Check





#Create Delta Table Using Various Methods:

#Method 1:

```
from delta.tables import *
DeltaTable.create(spark) \
        . tableName ("employee _ demo") \
        . addCoIumn("emp_ID" ,"INT") \
        . addCoIumn("emp_name" ,"STRING") \
        . addCoIumn("gender " ,"STRING") \
        . addCoIumn("salary" , "INT") \
        . addCoIumn( "Dept" ,"STRING") \
        . property ( "description" ,"table created for demo purpose") \
        . location( "/FileStore/tab1es/delta/createtabte") \
        . execute ( )
```

#Method 2:

```
%sql

CREATE TABLE IF NOT EXISTS employee_demo (
        emp_id INT,
        emp_Name STRING,
        gender STRING,
        salary INT,
        dept STRING,
) USING DELTA
Location '/FileStore/tab1es/delta/createtable'
```

#Method 3: Using Dataframe

```
employee_data = [(100, "Stephen" "M" 2000, "IT")
                 (200, "Phi lipp", "M", "HR")
                 (309, "Lara", "F", 6000, "SALES")]
employee _ schema = ["emp_id", "emp_name", "gender", "salary", "dept"]
df = spark.CreateDataFrame (data= employee_data, schema=employee _ schema)
display (df)
```

```
#Create table in the metastore using DataFrame's schema and write data to it
df.write.format(“delta").saveAsTable( "default.employee_demo”)

#read data from table
%sql

Select * from employee_demo
```

#5. Delta Table Instance

What is Delta Table Instance?

To make soft link(replica) to actual delta table

What is the Usage of it?

To perform DML operations using Pyspark language

```
#Create Delta Table Instance

#First Approach

deltainstance1= DeltaTable.forpath(spark,”/FileStore/tables/delta/createtable”)

#delta table to DataFrame.

display (deltainstance1.toDF( ))

#delete one record

deltainstance1.delete(“emp_id =200")

display (deltainstance1.toDF( ))

#Second Approach

deltainstance2 = DeltaTable.forName(spark, "employee_demo")

display (deltainstance2.toDF())
```

#How many ways Read CSV file in to DataFrame using PySpark

```
#Method 1:

# Initialize Spark session

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("CSVReadExample").getOrCreate()

# Read CSV file into DataFrame

Df1 = spark.read.csv("path/to/your/file.csv", header=True, inferSchema=True)

#Method 2:

df 2= spark.read.format("csv").load("/tmp/resources/zipcodes.csv")

//      or
```

df3 = spark.read.format("org.apache.spark.sql.csv").load("/tmp/resources/zipcodes.csv")

df3.printSchema()

#Method 3:

df4 = spark. read. option("header" , True). csv(" /tmp/resources/zipcodes . csv”)

#Method 4:

df3 = spark.read.options(delimiter=',').csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/zipcodes.csv")

#Method 5:

spark = SparkSession.builder().master("local[1]").appName("SparkByExamples.com").getOrCreate()

df = spark.read.csv("/tmp/resources/zipcodes.csv")

df.printSchema()

#Using Header Record For Column Names

df2 = spark.read.option("header",True).csv("/tmp/resources/zipcodes.csv")

#Read Multiple CSV Files

df = spark.read.csv("path1, path2, path3")

#Read all CSV Files in a Directory

df = spark.read.csv("Folder path")

 # Infer-Schema, delimiter, header

df4 = spark.read.options(inferSchema='True', delimiter=',')\.csv("src/main/resources/zipcodes.csv")

df4 = spark.read.option("inferSchema",True).option("delimiter",",")\.csv("src/main/resources/zipcodes.csv")

df3 = spark.read.options(header='True', inferSchema='True', delimiter=',')\.csv("/tmp/resources/zipcodes.csv")

df3 = spark.read.options(header='True', inferSchema='True', delimiter=',')\.csv("/tmp/resources/zipcodes.csv")

#Hive Tables (read & write)

#read DataFrame from hive using PySpark

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("ReadFromHiveExample") \
    .config("hive.metastore.uris", "thrift://<metastore-host>:<metastore-port>") \
    .enableHiveSupport() \
```

```
    .getOrCreate()
table_name = "your_database.your_table"
df.show()
spark.stop()
```

#Write DataFrame into hive using PySpark

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("WriteToHiveExample").enableHiveSupport().getOrCreate()
df = ...
df.write.saveAsTable("database_name.table_name")
```

#1. PySpark Write to CSV File

```
# Syntax of DataFrameWriter.csv()
DataFrameWriter.csv(path, mode=None, compression=None, sep=None, quote=None, escape=None, header=None, nullValue=None, escapeQuotes=None, quoteAll=None, dateFormat=None, timestampFormat=None, ignoreLeadingWhiteSpace=None, ignoreTrailingWhiteSpace=None, charToEscapeQuoteEscaping=None, encoding=None, emptyValue=None, lineSep=None)
```

#2. Write PySpark to CSV file

```
#Write DataFrame to CSV file
df.write.csv("/tmp/spark_output/zipcodes")
```

#3. PySpark Write to CSV with Header

```
# Write CSV file with column header (column names)
df.write.option("header",True).csv("/tmp/spark_output/zipcodes")
```

#4. Write CSV Options

```
# Other CSV options
df2.write.options(header='True', delimiter=',').csv("/tmp/spark_output/zipcodes")
```

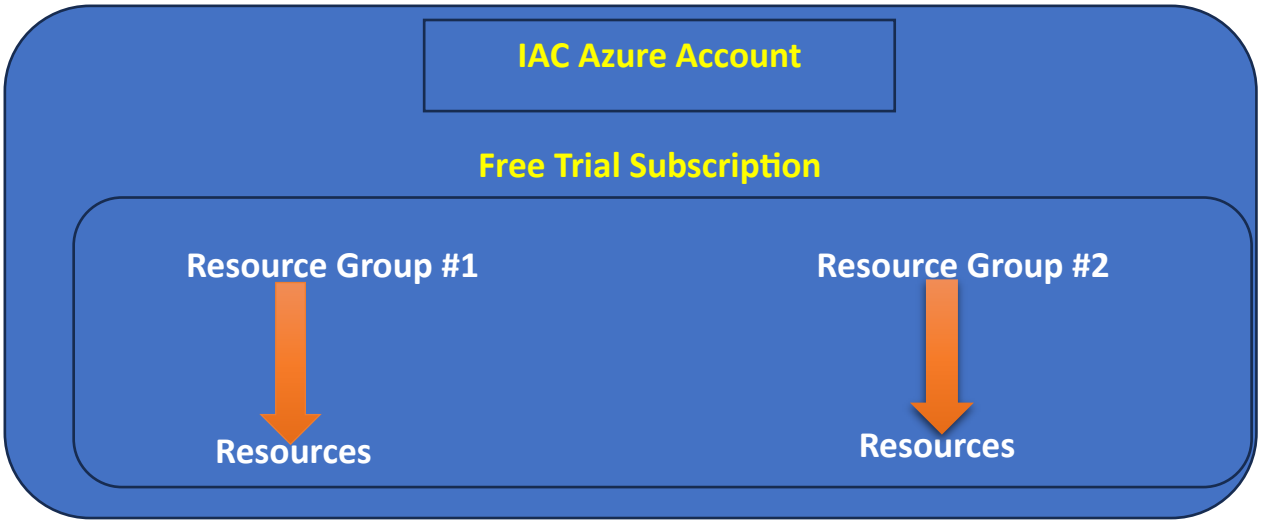
#5. Write Saving modes

PySpark DataFrameWriter also has a method mode() to specify saving mode.

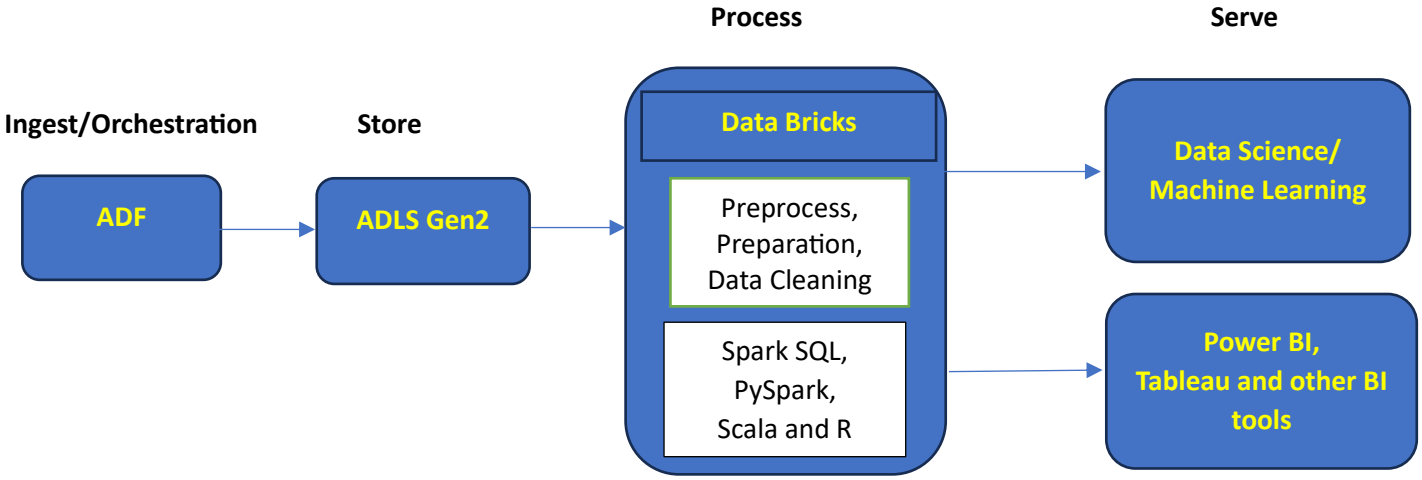
overwrite – mode is used to overwrite the existing file.
append – To add the data to the existing file.
ignore – Ignores write operation when the file already exists.
error – This is a default option when the file already exists, it returns an error.

```
df2.write.mode('overwrite').csv("/tmp/spark_output/zipcodes")
    # You can also use this
df2.write.format("csv").mode('overwrite').save("/tmp/spark_output/zipcodes")
```

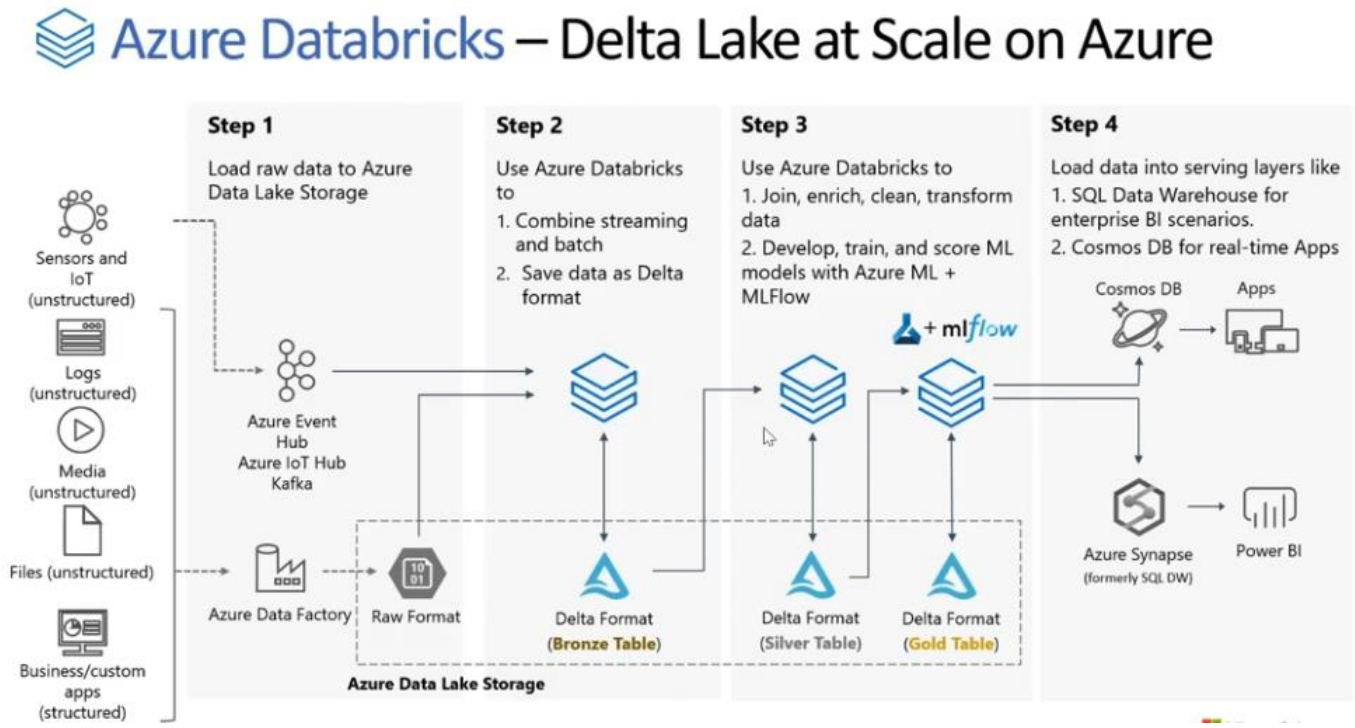
Your AZURE Account Structure (Tenant)

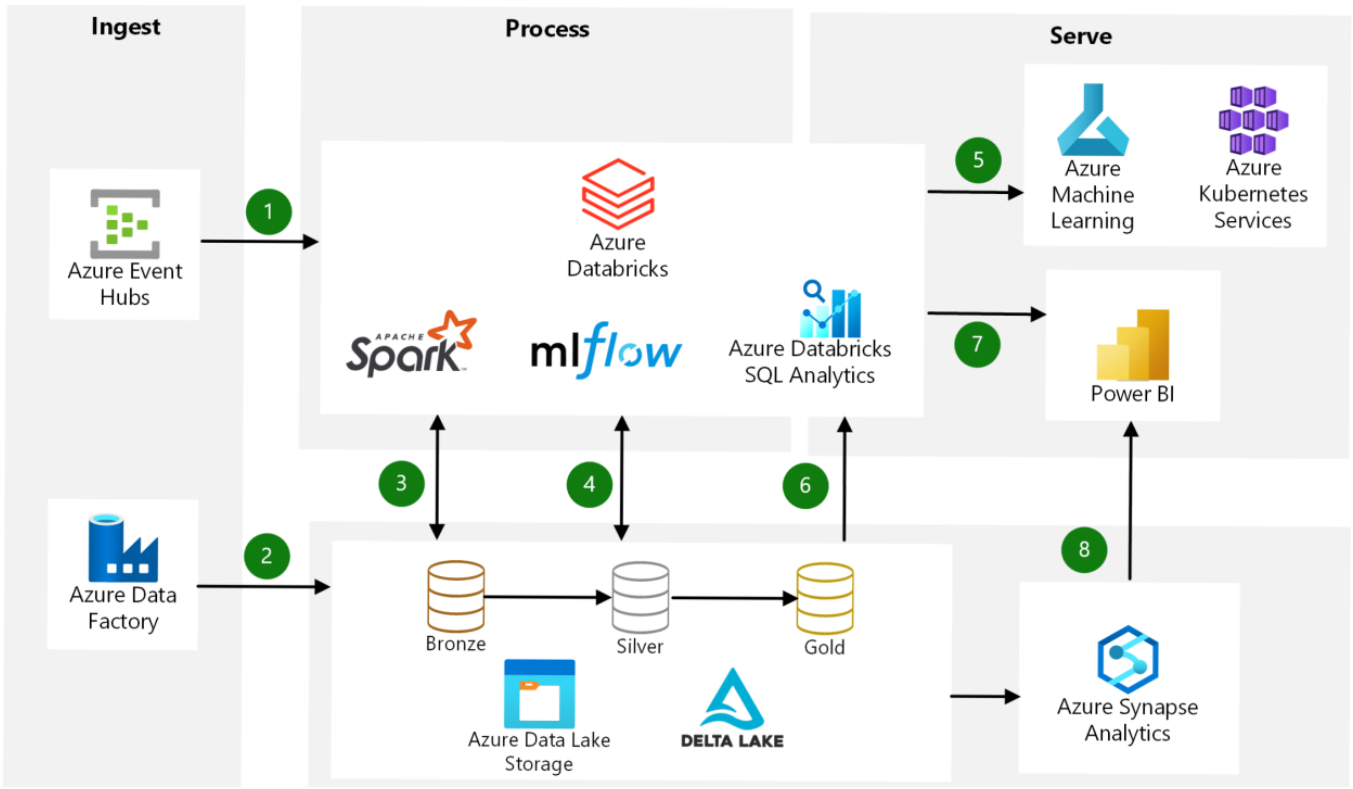


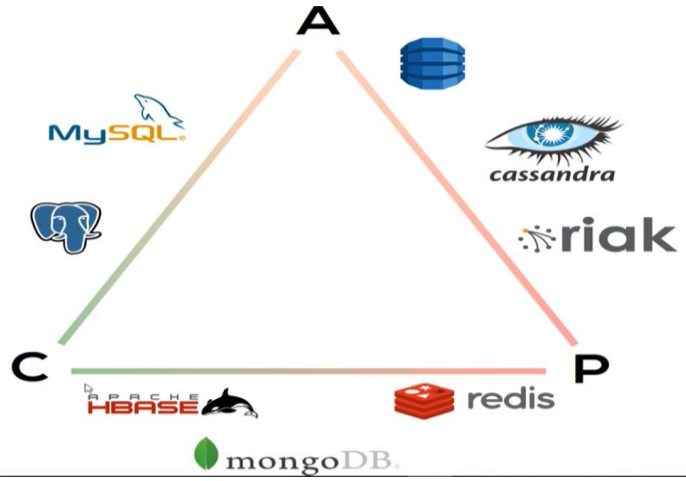
Databricks Architecture

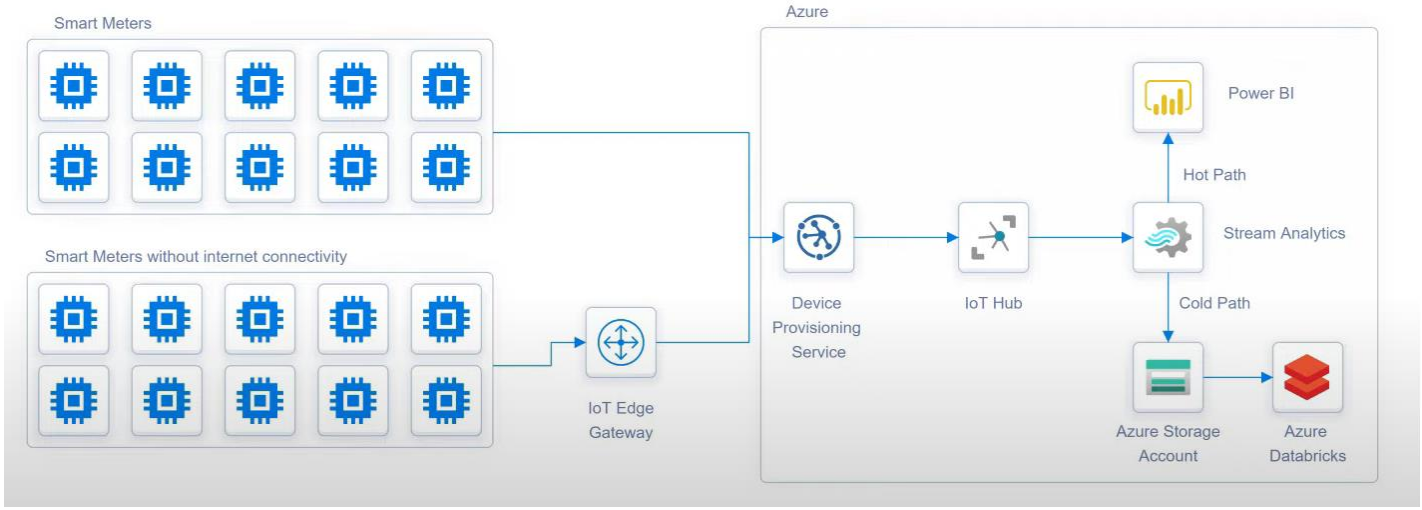


Mounting a *file system binds it to a directory (mount point) and makes it accessible to the system*. The file system root (/) is always mounted.




Ingest
Process
Serve
Azure Event Hubs
Azure Data Factory
Azure Databricks
Spark
mlflow
Azure Databricks SQL Analytics
Azure Machine Learning
Azure Kubernetes Services
Power BI
Azure Synapse Analytics
Bronze
Silver
Gold
Azure Data Lake Storage
DELTA LAKE
1
2
3
4
5
6
7
8


A
C
P
MySQL
cassandra
riak
redis
mongoDB.


Smart Meters
Smart Meters without internet connectivity
IoT Edge Gateway
Azure
Device Provisioning Service
IoT Hub
Power BI
Hot Path
Stream Analytics
Cold Path
Azure Storage Account
Azure Databricks

Load Patterns

1. Truncate and Load
2. Merge Load
3. Incremental Load
4. Bulk Table Transfer

DATA LAKE:
Data Lake is a storage repository that cheaply stores a vast amount of raw data in its native format. It Consists of current and historical data dumps in various formats including XML, JSON, CSV, Parquet, etc.

Drawbacks in Data Lake:

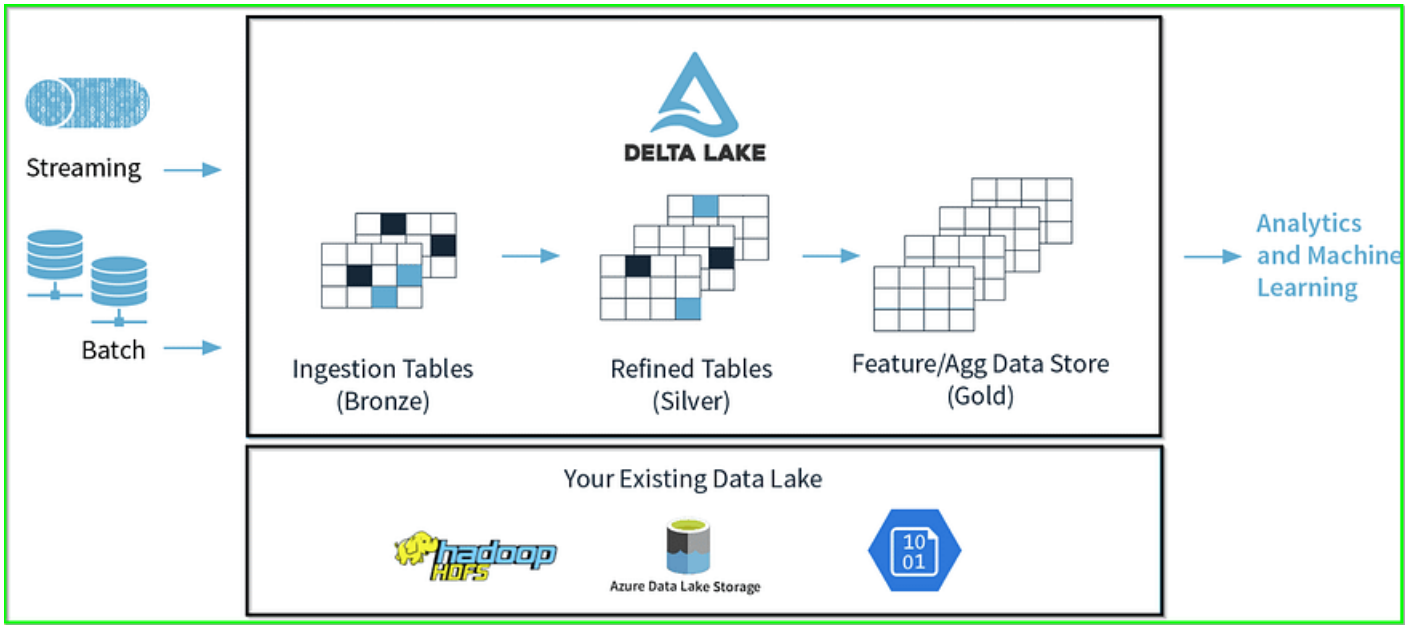
- Doesn't provide Atomicity — No all or nothing, it may end up storing corrupt data.
- No Quality Enforcement — It creates inconsistent and unusable data.
- No Consistency/Isolation — It's impossible to read and append when there is an update going on.

DELTA LAKE:

Delta Lake allows us to incrementally improve the quality until it is ready for consumption. Data flows like water in Delta Lake from one stage to another stage (Bronze ---> Silver ---> Gold).

- Delta lake brings full ACID transactions to Apache Spark. That means jobs will either complete or not at all.
- Delta is open-sourced by Apache. You can store a large amount of data without worrying about locking.
- Delta lake is deeply powdered by Apache Spark which means that the Spark jobs (batch/stream) can be converted without writing those from scratch.

Delta Lake Architecture:



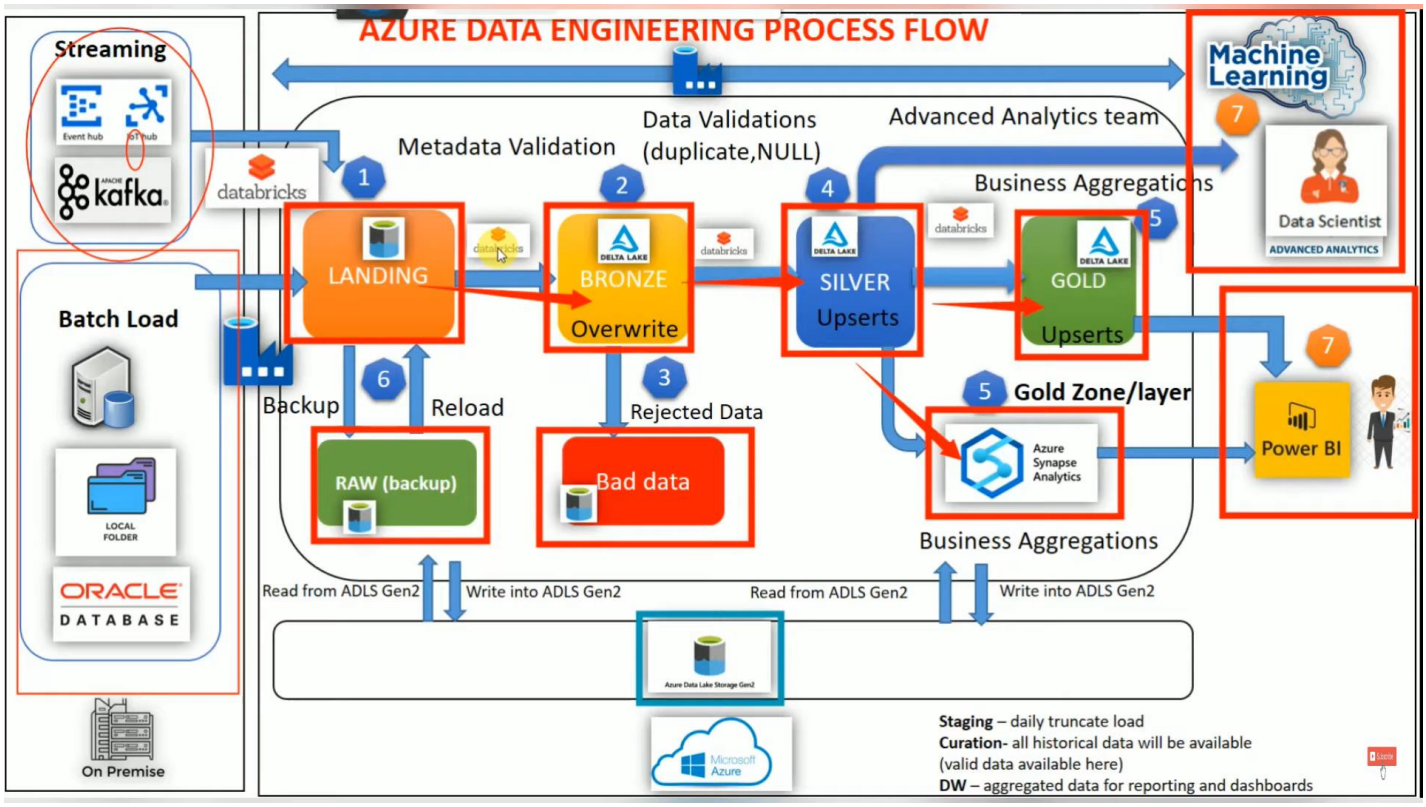
Bronze Tables:
Data may come from various sources which could be Dirty. Thus, it is a dumping ground for raw data.

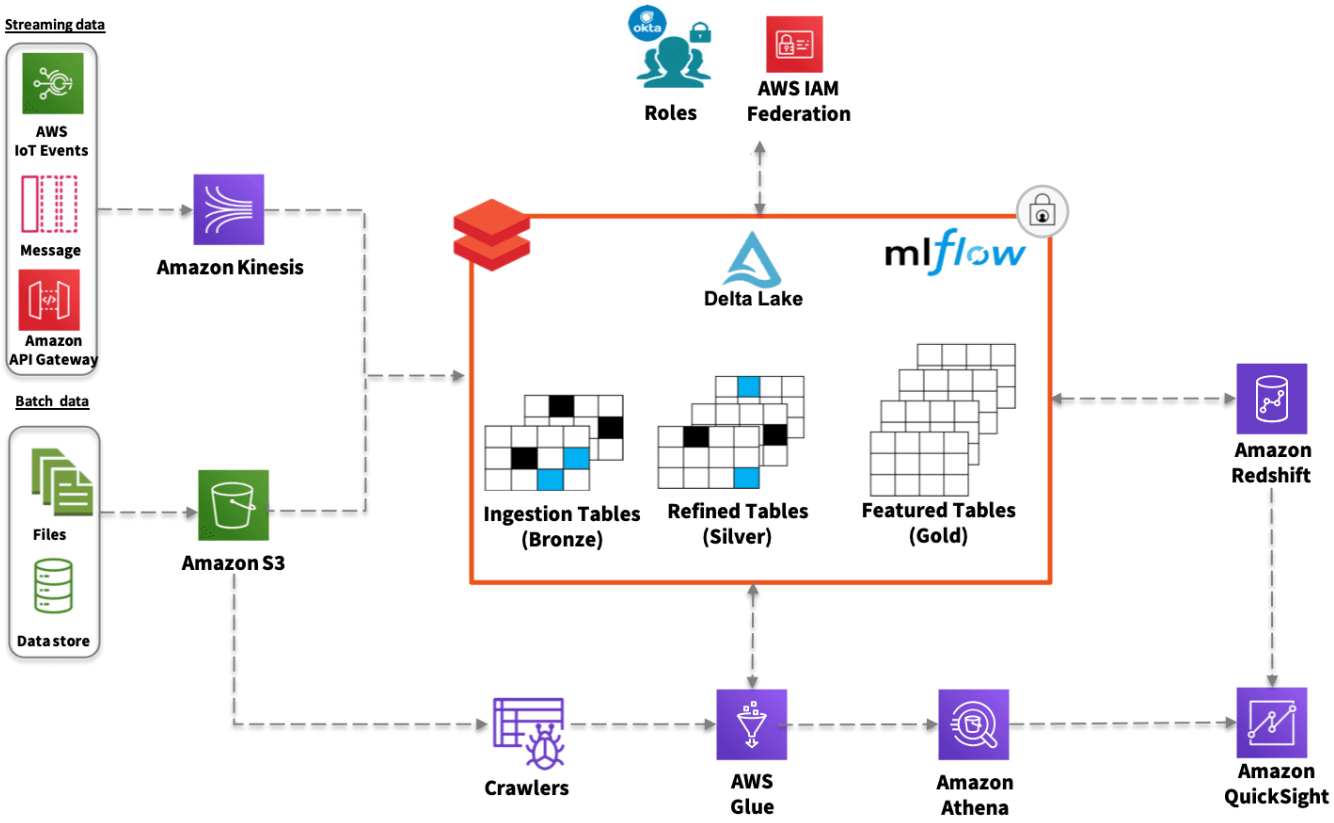
Silver Tables:

Consists of Intermediate data with some cleanup applied.
It is Queryable for easy debugging.

Gold Tables:

Consists of clean data, which is ready for consumption.





Pipeline Parameter vs. Variable in Azure Data Factory

In Azure Data Factory (ADF), both parameters and variables are used to manage and control the behavior of your data pipelines. However, they serve different purposes and are used in different contexts. Let's explore the differences between pipeline parameters and pipeline variables:

1. **Pipeline Parameters:**
   - Parameters are used to provide dynamic input values to a pipeline at runtime.
   - Parameters are defined at the pipeline level and can be used to pass values to the pipeline from external sources, such as trigger parameters, pipeline parameters, or even from other pipelines.
   - Parameters are typically used for values that might change between pipeline executions, such as file paths, database connection strings, or filtering criteria.
   - Parameters are defined when you create or configure a trigger that initiates the pipeline run. They can also be set when you manually trigger the pipeline run.
   - Parameters are read-only within the pipeline. They cannot be modified or changed during the pipeline execution.
2. **Pipeline Variables:**
   - Variables are used to store and manage temporary values within a pipeline.
   - Variables are defined at the pipeline level and can be used to store values that might change or need to be manipulated during the pipeline execution.
   - Variables are useful for tasks like counters, loop control, logging, or transformations.
   - Variables can be set, updated, and modified within the pipeline using activities or expressions.
   - Unlike parameters, variables are mutable within the pipeline, which means you can change their values during the pipeline run.

In summary, while both pipeline parameters and variables are used for controlling and customizing pipeline behavior, they serve different purposes. Parameters are primarily used for passing external input values to a

pipeline, while variables are used for storing temporary data or values that may change during the pipeline execution. The choice between using parameters or variables depends on the specific requirements of your data pipeline and how you need to manage and control the flow and behavior of your data processing.

It's important to note that Azure Data Factory supports parameterized pipelines, where pipeline parameters can be used to control the execution of nested pipelines or to provide dynamic values to activities within the pipeline. This allows for greater flexibility and reusability when designing complex data workflows.

2. azure data factory expression language

Azure Data Factory (ADF) provides its own expression language that you can use to create dynamic expressions and transformations within your data pipelines. This expression language is used in various places within ADF, such as defining parameters, variables, dataset properties, activity settings, and more. The expression language allows you to manipulate and transform data, control workflow logic, and work with metadata.

Here are some key aspects of the Azure Data Factory expression language:

1. **Functions and Operators:** ADF expression language supports a wide range of functions and operators for performing operations like string manipulation, mathematical calculations, date/time operations, conditional logic, and more.
2. **Variables and Parameters:** You can use expressions to work with pipeline parameters and variables. For example, you can reference parameter values in activity settings or use variables to store and manipulate temporary data.
3. **Dynamic Content:** Expressions allow you to create dynamic content within various fields. For example, you can dynamically generate file paths, dataset names, or other properties based on runtime values.
4. **Accessing Metadata:** The expression language enables you to access metadata about your data, such as column names, data types, and other dataset properties. This can be useful for building dynamic transformations and mappings.
5. **ForEach Loops:** You can use expressions to define the iteration behavior of ForEach loops, including specifying the items to iterate over and dynamically constructing output paths.
6. **Conditional Logic:** Expressions support conditional statements and functions (e.g., `if`, `coalesce`) that allow you to control the flow of your data pipelines based on specific conditions.
7. **Type Conversion:** The expression language allows you to perform type conversions and handle different data types, which is particularly useful for mapping and transformations.
8. **Data Movement and Transformation:** You can use expressions in various ADF activities (e.g., Copy Data, Data Flow) to define transformations, source and sink settings, and mapping rules.

Azure Services

Storage

Web

Database

IOT

Big Data

AI

Azure IoT Central

Compute

Networking

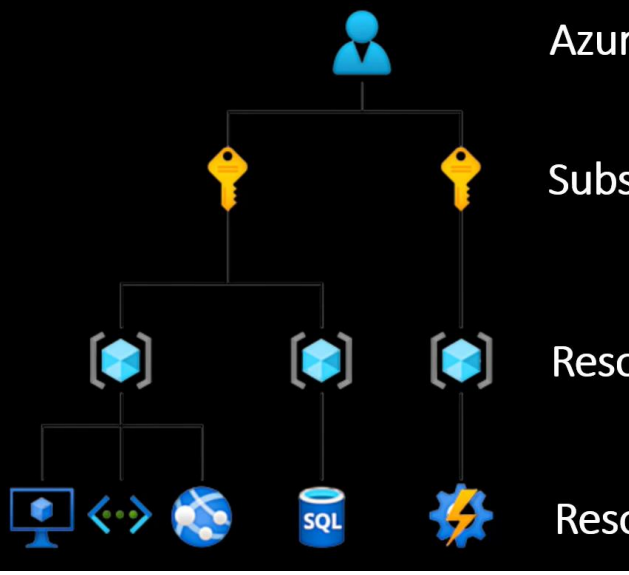
Monitoring

Azure Accounts

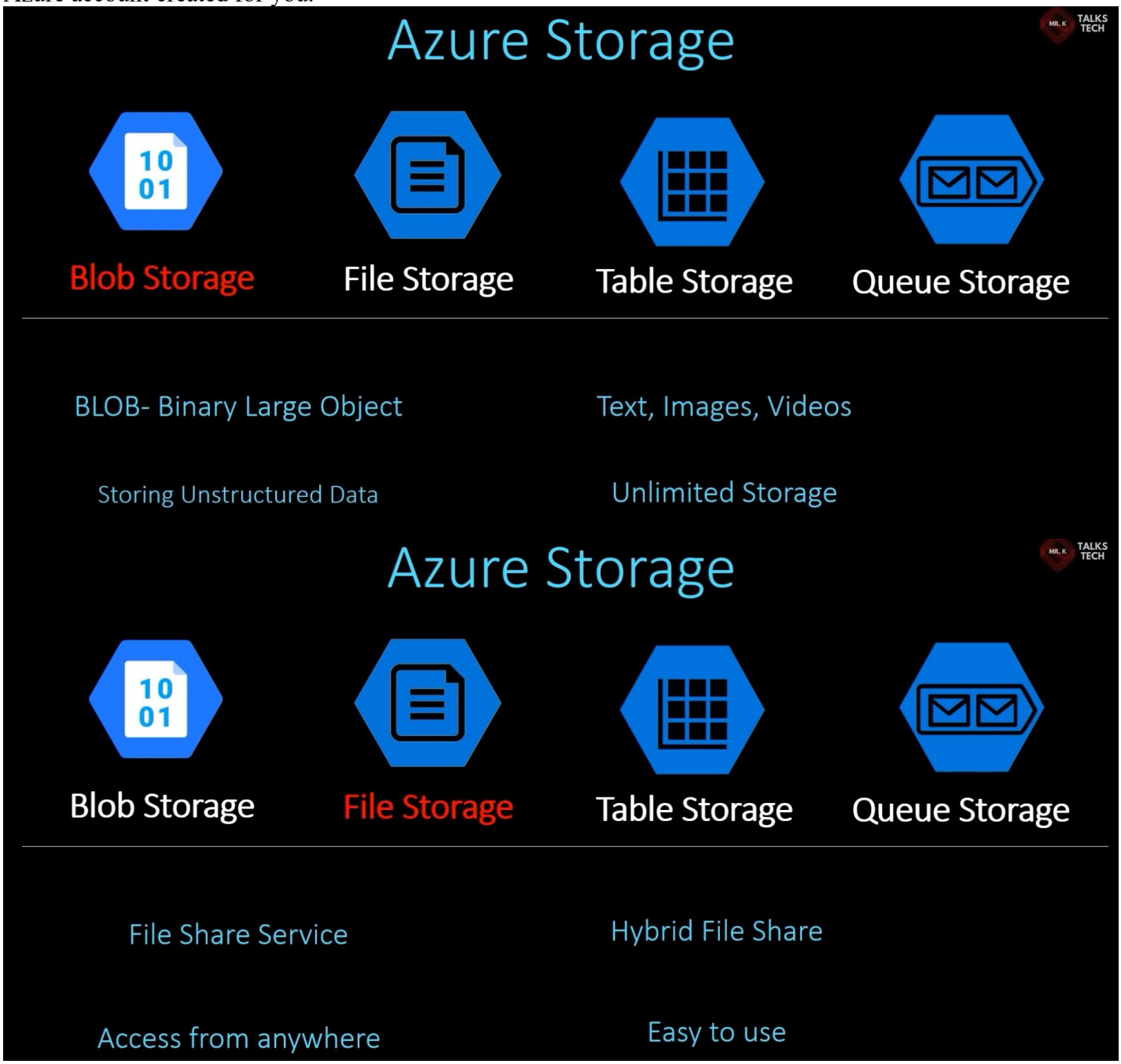
Tenant

Subscription

Resource Group

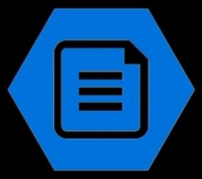


Azure Active Directory: AAD is not a resource like **Azure Data Factory** whenever create Azure account created for you.



Azure Storage

Blob Storage



File Storage

Table Storage

Queue Storage

No SQL Key-Value Storage

Less cost compared to traditional SQL Table

No Fixed Schema

High Access Speed

Azure Storage

Blob Storage

File Storage

Table Storage

Queue Storage

Stores data in First-In First-out rule

Can access the messages from anywhere

Helps in storing large number of messages

Supports REST API

Azure Storage

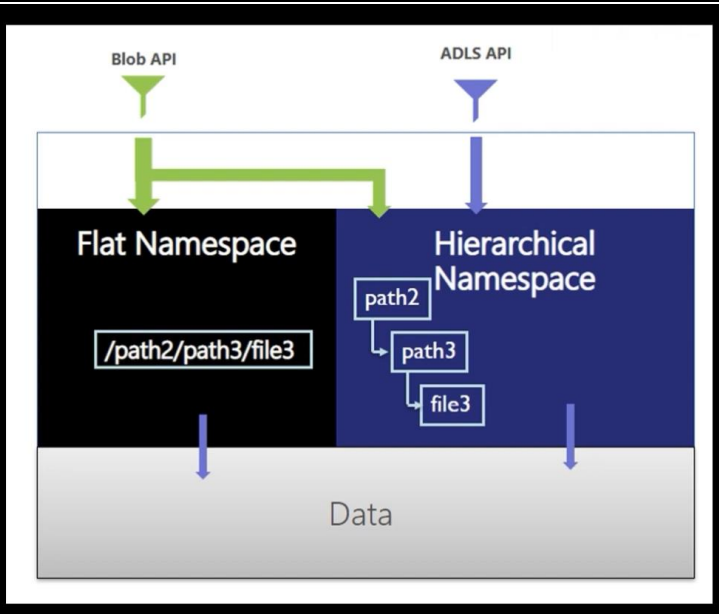
Blob Storage

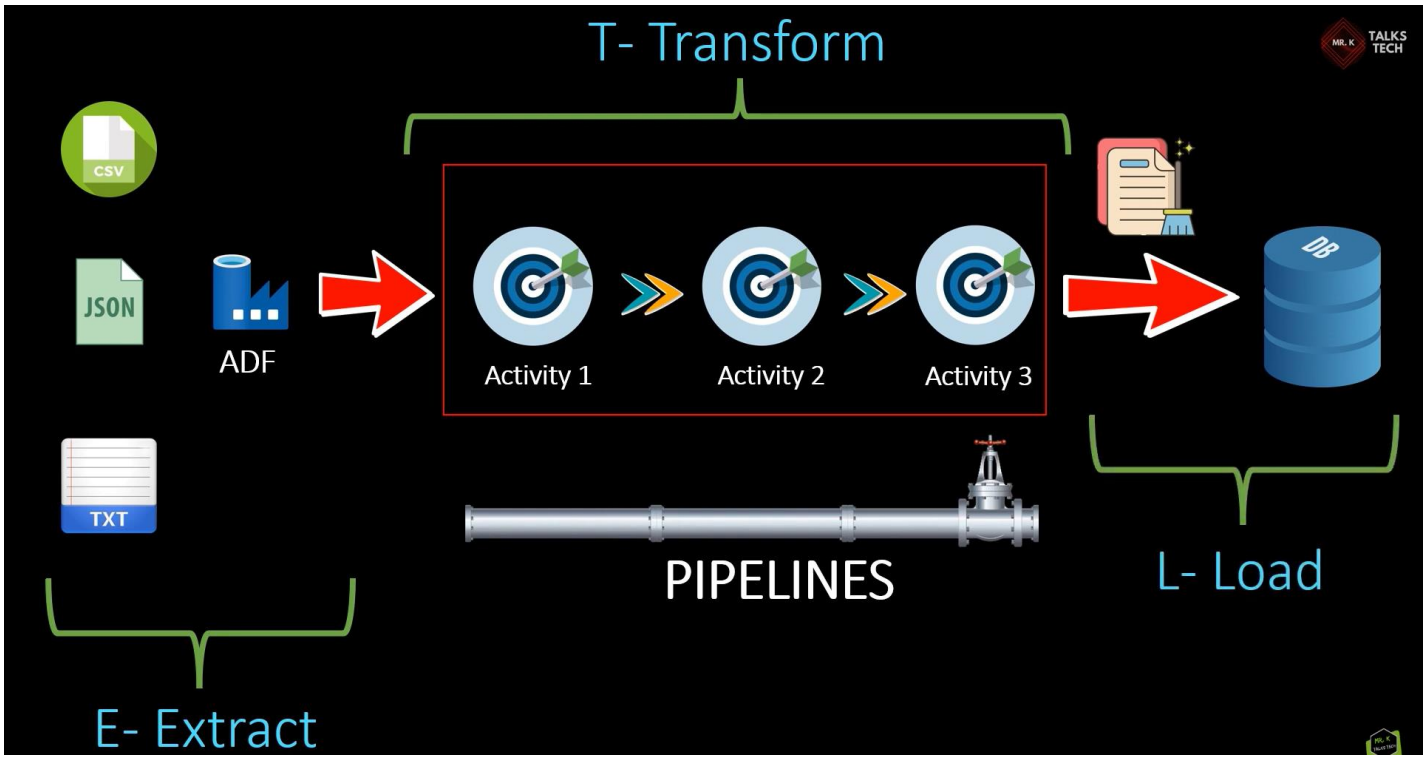
File Storage

Table Storage

Queue Storage

Hierarchical Namespace = Azure Data Lake Gen2





Azure Data Factory:

Home

*****Author**

- Factory Resources
- Pipelines
- Change data capture (preview)
- Datasets
- Data flows
- Power Query

Monitor

Dashboards

Runs

- Pipeline runs
- Trigger runs
- Change data capture (preview)

Runtimes & Sessions

- Integration runtimes
- Data flow debug

Notifications

- Alerts & metrics

Manage

General

- Factory settings

Connection

- Link services
- Integration runtimes
- Microsoft Purview

Source Control

- Git configuration
- ARM (Azure Resource Manager) Template

Author

- Triggers
- Global parameters
- Data flow libraries

Security

Credentials
Customer managed key
Manged private endpoints

Learning Center

Quick Start
Tutorials
Videos
Template Gallery

Integration Runtime → **Linked Service** → **Datasets**

Integration Runtime

Compute Infrastructure for Data Integration Three Types

- Azure IR
- Self-hosted IR
- SSIS IR

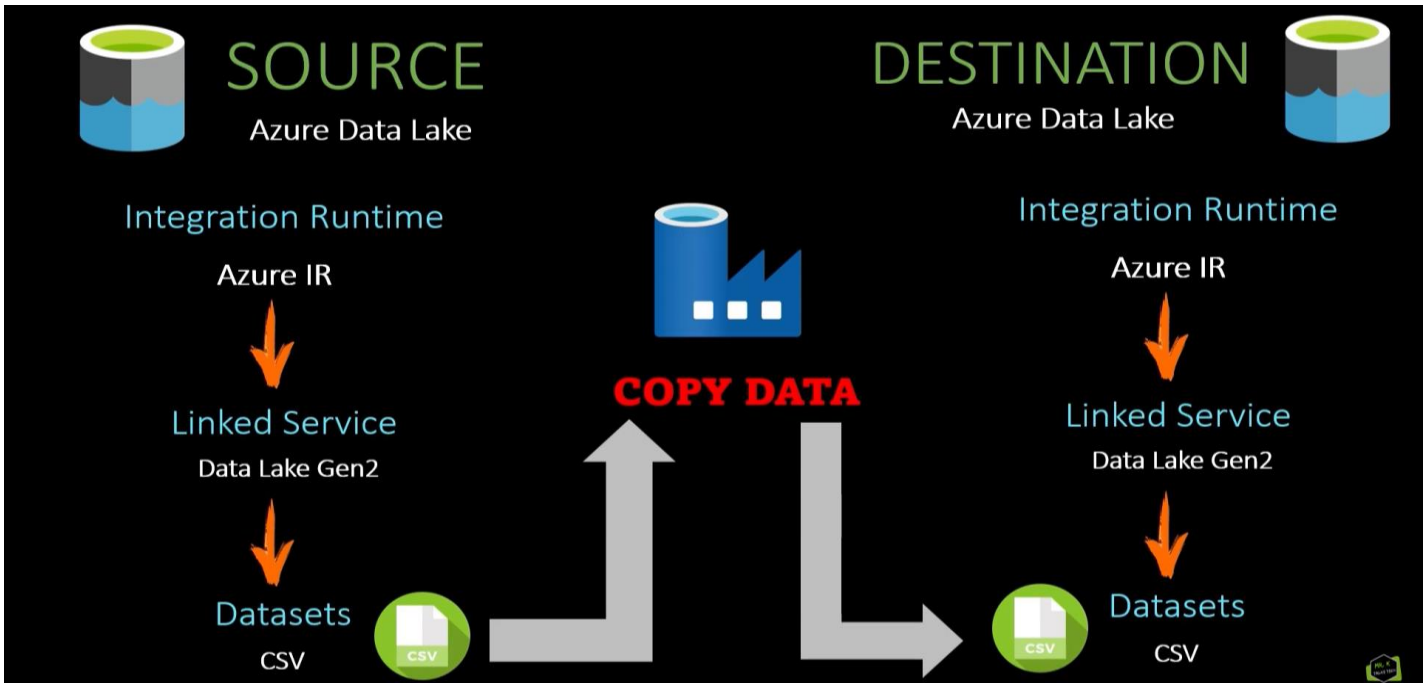
Linked Service

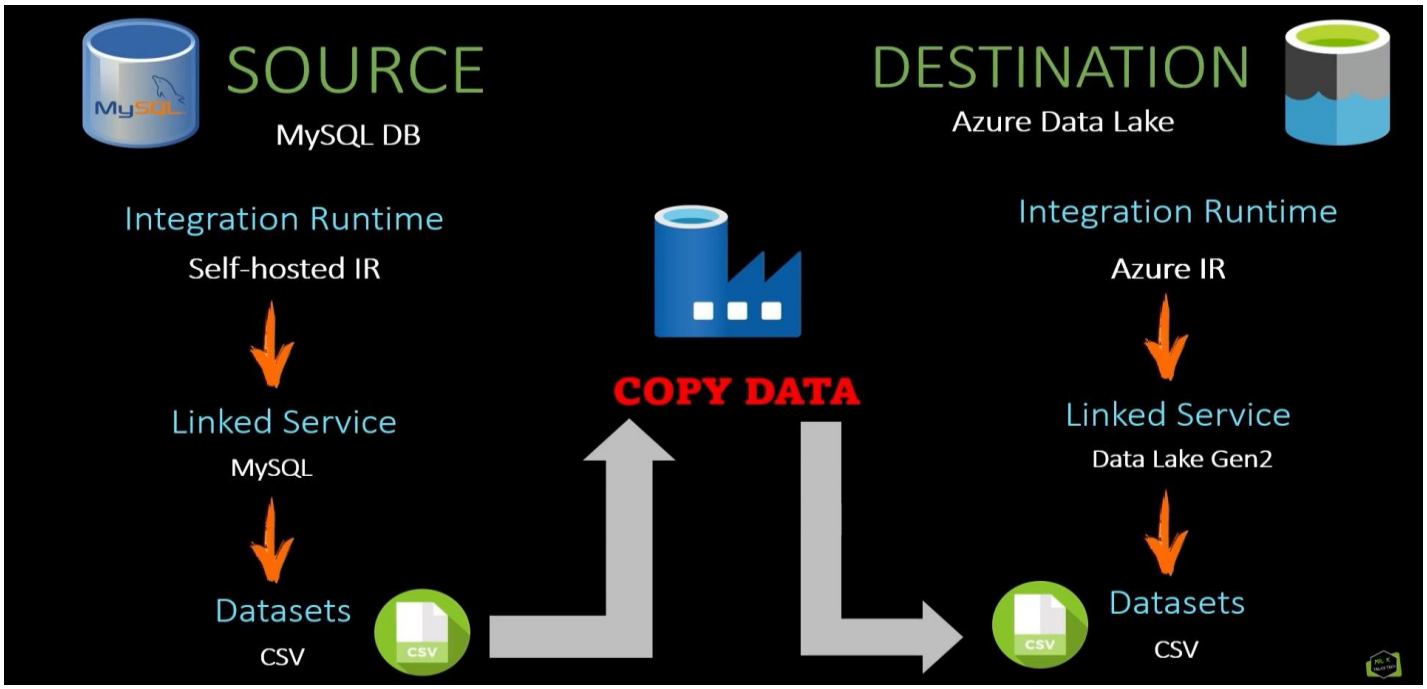
- It is much like connection strings, which define the connection information needed for the ADF to connect to the data source.
- More than 85 in-built linked service connectors are available inside ADF.
- You need an Integration runtime to create a linked service connection.

Datasets

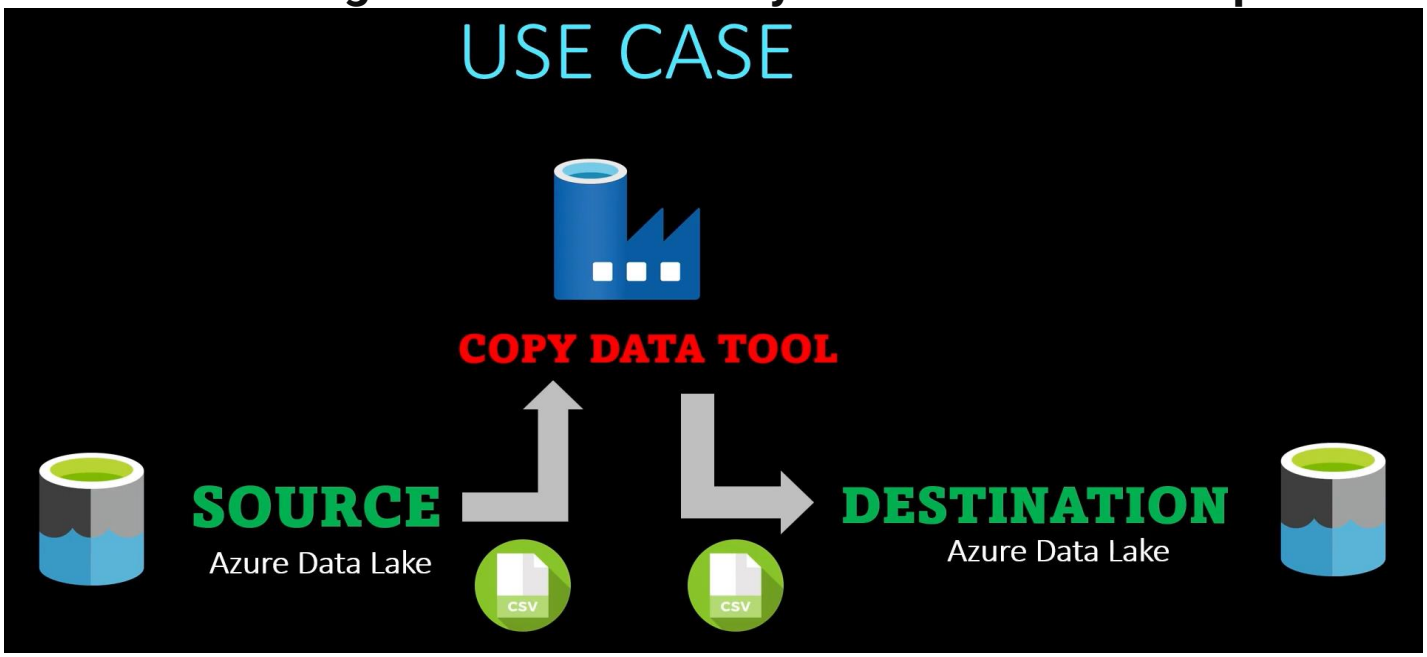
- It is the structure/ format of the data.
- You need to have a linked service connection to create a Dataset.

Avro
Excel
Parquet
Binary
JSON
XML
Delimited-Text
ORC





COPY Data using Azure Data Factory COPY DATA TOOL option



1. Properties
2. Source

   Dataset

   Configuration
3. Destination
4. Settings
5. Review and finish

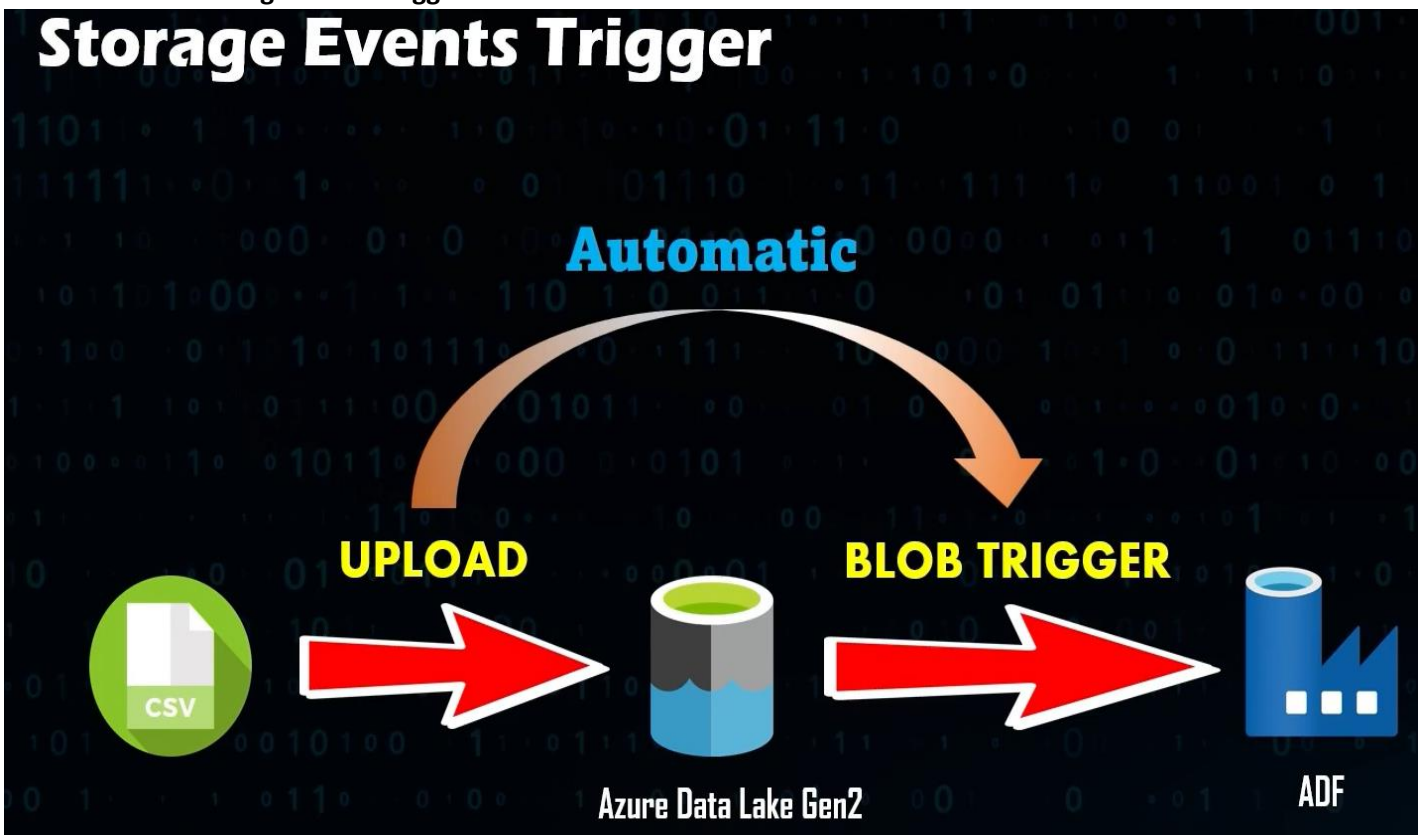
Author--->click on +--->Copy Data Tool--->Built-in copy task--->Task cadence (run once)

Types of Triggers

- **Scheduled Trigger.**
  - ✓ This trigger runs a pipeline on a specific schedule, such as hourly, daily, weekly, or monthly.

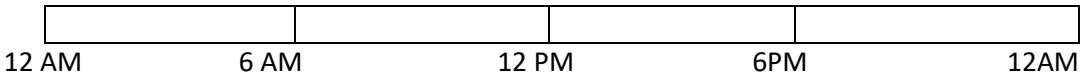
- ✓ Schedule triggers can be created and configured using the ADF portal.

➢ **Storage Events Trigger.**



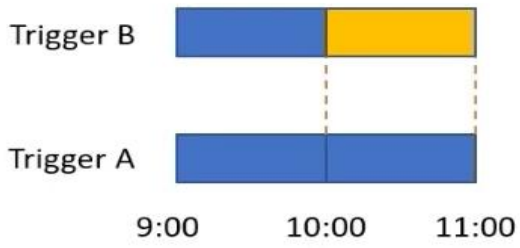
➢ **Tumbling Window Trigger.**

- ✓ This trigger runs a pipeline on a periodic time interval from a specified start time
- ✓ Example: Setting up the tumbling trigger to run every 6 hours in a day

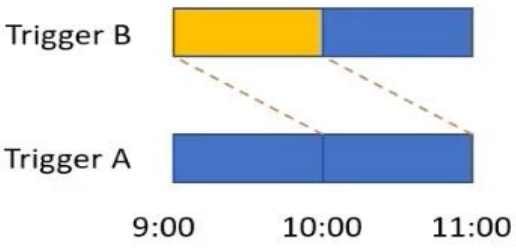


➢ **Custom Triggers**

Dependency offset

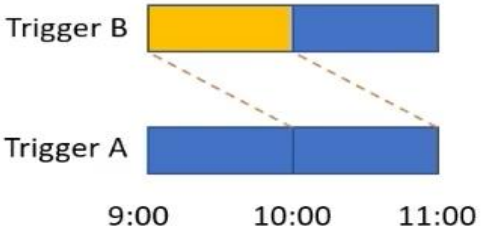


Dependency: A -> B
Offset: 0

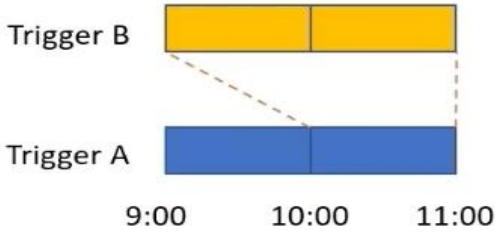


Dependency: A -> B
Offset: -1 hour

Dependency size

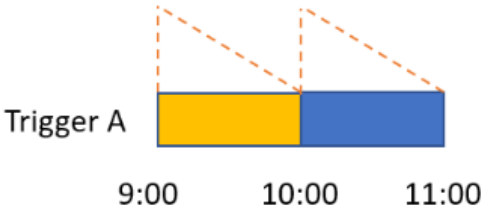


Dependency: A -> B
Offset: -1 hour
Size: unspecified

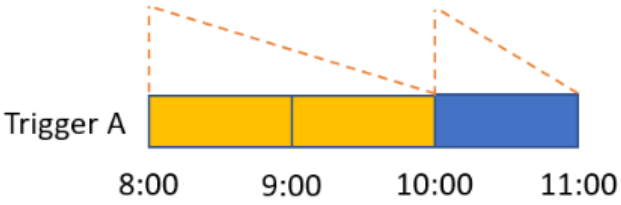


Dependency: A -> B
Offset: -1 hour
Size: 2 hours

Self-dependency



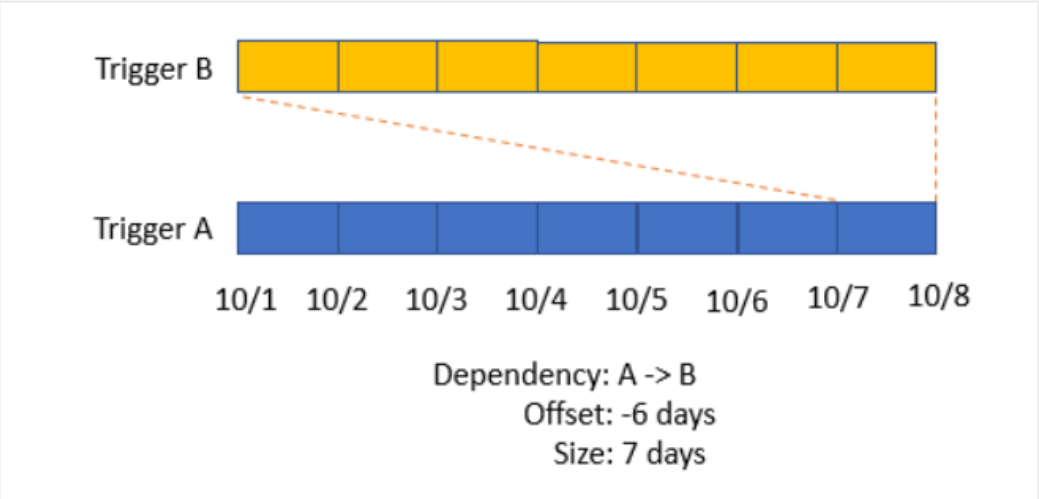
Dependency: A -> A
Offset: -1 hour
Size: unspecified



Dependency: A -> A
Offset: -2 hour
Size: 2 hours

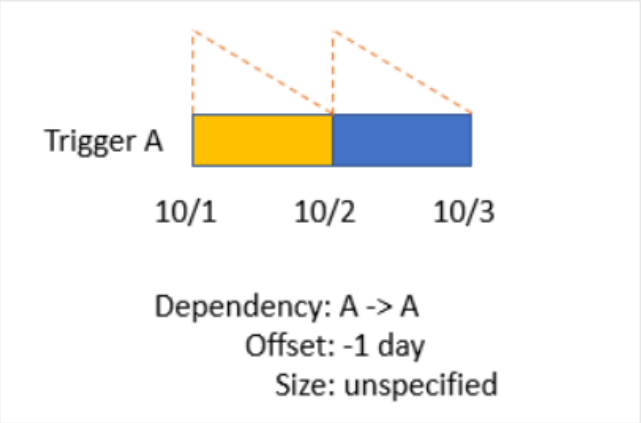
Dependency on another tumbling window trigger

A daily telemetry processing job depending on another daily job aggregating the last seven days output and generates seven day rolling window streams:



Dependency on itself

A daily job with no gaps in the output streams of the job:



has to work on past?

Db1

Tbl1——> start time endtime past

Tbl1

need to have dependency?

Trigger1

Trigger2

need concurrency?

need to access actual starttime and endtime?

triggerOutputs ( ).WindowStartTime

triggerOutputs ( ).WindowEndTime

Why Scheduled Trigger less reliable than Tumbling Windows Trigger?

- Tumbling window triggers are more reliable because of retry capability as well as because they can retain state.

12. Integration runtime in Azure Data Factory

Integration Runtime in ADF

• The Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory to provide the following data integration capabilities across different network environments:

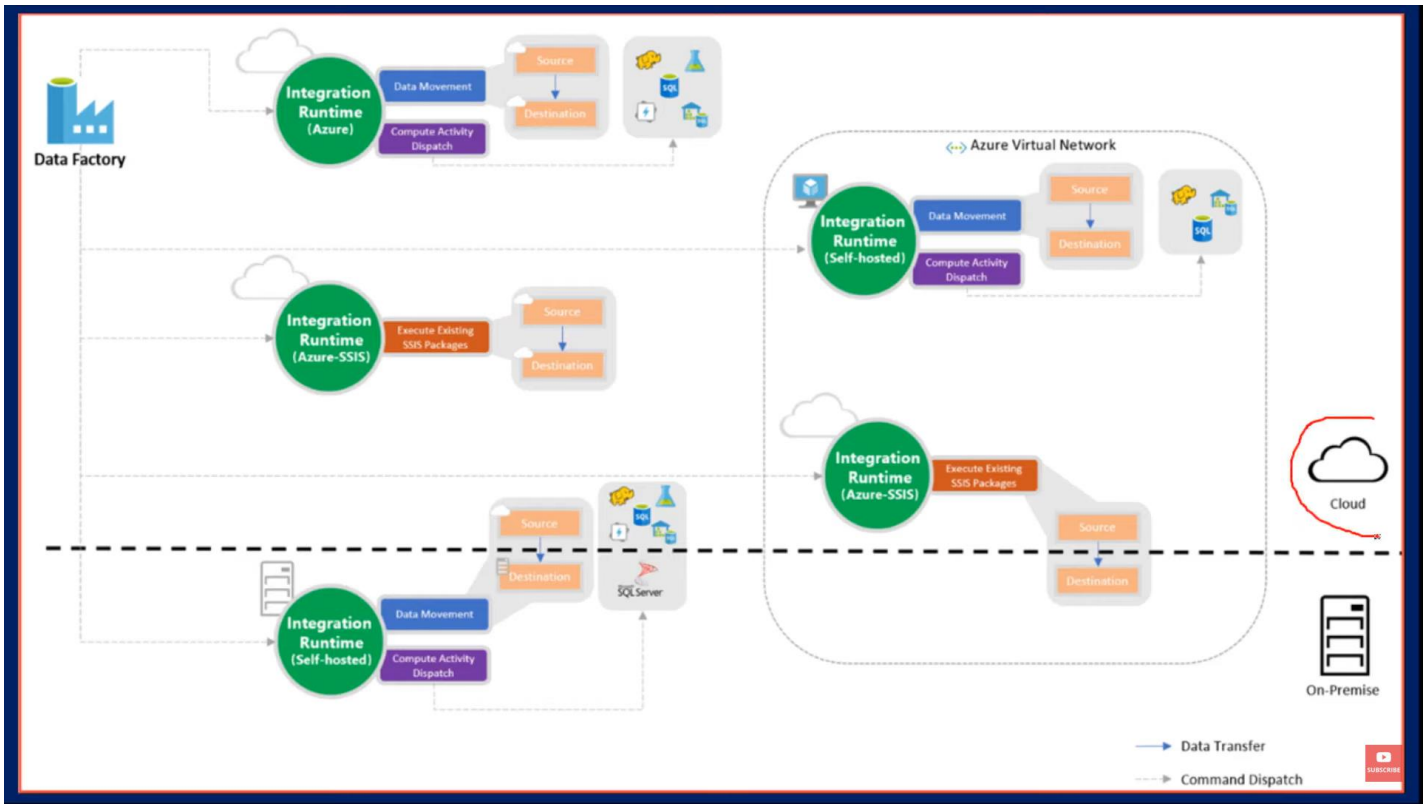
Data Flow: Execute a Data Flow in managed Azure compute environment
Data Movement: Copy data across data stores in public network a private network
Activity Dispatch: Dispatch and monitor transformation activities running on a variety of compute services.
SSIS Package Execution: Execute SQL Server Integration Services (SSIS) packages in a managed Azure compute environment.

Linked service - Target resource data store or compute service.
Activities - Task which u want to perform
integration runtime as bridge - Linkedservice and Activity

Types of Integration Runtimes

Data Factory offers three types of Integration Runtime, and you should choose the type that best serve the data integration capabilities and network environment needs you are looking for. These three types are:

IR type	Public network	Private network
Azure	Data Flow Data movement Activity dispatch	
Self-Hosted	Data movement Activity dispatch	Data movement Activity dispatch
Azure-SSIS	SSIS package execution	SSIS package execution



13. Azure Integration runtime in Azure Data Factory

Azure Integration runtime
• Azure integration runtime is capable of performing below in public network.
*Running DataFlows
*Running Copy Activities between Cloud data stores
*Running Transform Activities
• Azure Integrating runtime supports connecting to data stores and compute services with public accessible endpoints.
• Azure integration runtime provides a fully managed, serverless compute in Azure
• Azure IR is elastically scaled up accordingly without you having to explicitly adjusting size of the Azure Integration Runtime.

Azure Integration runtime

• Azure integration runtime will come by default with location as auto- resolve.

Create Azure Integration runtime

• You only need to explicitly create an Azure IR when you would like to explicitly define the location of the IR, or if you would like to virtually group the activity executions on different IRs for management purpose.

14. Self-Hosted Integration runtime in Azure Data Factory

Self-Hosted Integration runtime.

• Self-Hosted Integration runtime is capable

- Performing data movement activities between cloud data stores and in private network

• Running transform activities against compute resources in on-premises or Azure virtual network

• Self-hosted IR needs to be installed on an on-premises machine or a virtual machine inside a private network. Currently, its supports running the self-hosted IR on a Windows operating system.

15. Setting up Self Hosted Integration runtime in Azure Data Factory

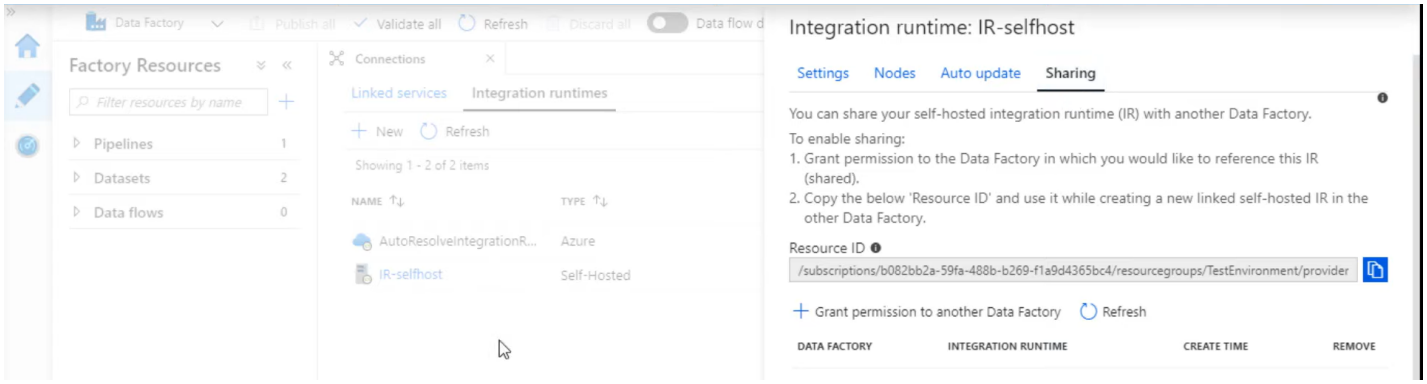
IR: used by ADF to perform its activities
Self-host IR: Public and Private
Cloud storage
On-prem storage

High Availability

• You can associate a self-hosted integration runtime with multiple on- premises machines or virtual machines in Azure. These machines are called nodes.

• You can have up to four nodes associated with a self-hosted integration runtime.

• Benefits of having multiple nodes is High availability and improved performance and throughput during data movement.

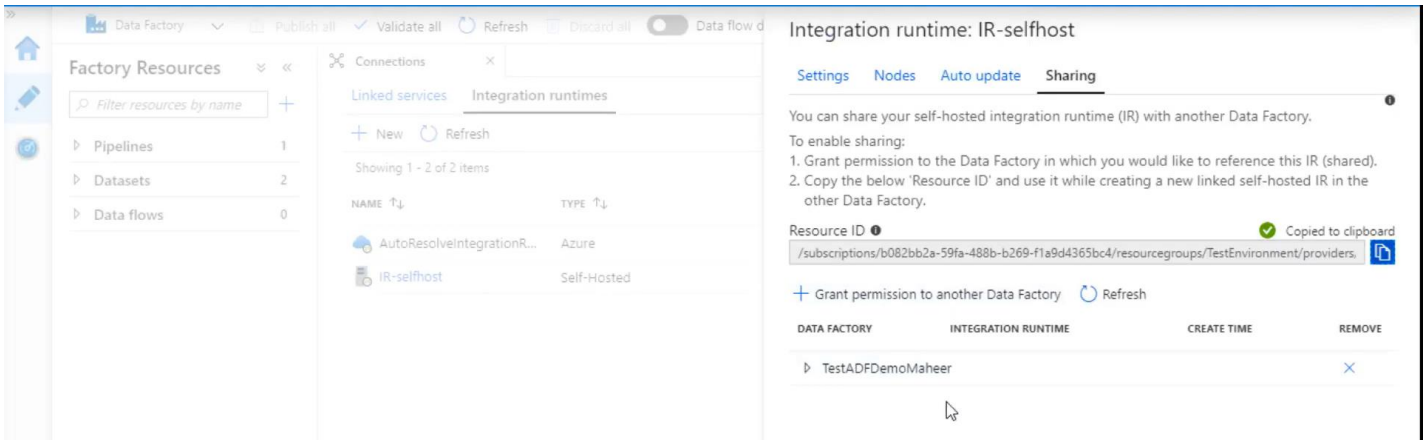


16. Shared Self Hosted Integration runtime in Azure Data Factory

IR: ADF.
Data integration.
Self -hosted IR:
On-prem - cloud
Setting self-hosted setup a Software.

Self IR for every ADF.

Now option with self IR



17. Parameterize Linked Services in Azure Data Factory

- You can parameterize a **linked service** and pass dynamic values at run time.

SQL server:
10 DBs
creating IO linked services for all 10 DBs.
Parameterization....
only one liked service.

```
@(linkedserices().dbName)
```

18. Parameterize Datasets in Azure Data Factory

- You can parameterize a **dataset** and pass dynamic values at run time.

```
@(dataset().tableName
```

19. Parameterize Pipelines in Azure Data Factory

copy data from one
Tablel and Table2
SQL table to another SQL Table

- ✓ Linked services are much like connection strings, which define the connection information needed for the service to connect to external resources.
- ✓ The dataset represents the structure of the data within the linked data stores, and the linked service defines the connection to the data source.
- ✓ Create pipeline---> add copy activity--->click outside copy activity

General	Parameters	Variables	Output
	New +	New +	

20. System Variables in ADF

System Variables are available at below three scopes. We can use these variables in expressions in Azure data factory.

Pipeline Scope System Variables:

Copy data from one storage to another and then log details of pipeline execution in to SQL DB.

Schedule trigger Scope System Variables:

Copy data one storage to another storage daily for the given date.

Tumbling Windows Trigger Scope System Variables:

Copy data one storage to another storage for every hour.

- **Pipeline Scope**

Variable Name	Description
@pipeline().DataFactory	Name of the data or Synapse workspace the pipeline run is running in
@pipeline().Pipeline	Name of the pipeline
@pipeline().RunId	ID of the specific pipeline run
@pipeline().TriggerType	The type of trigger that invoked the pipeline (for example, `ScheduleTrigger`, `BlobEventsTrigger`). For a list of supported trigger types, see Pipeline execution and triggers. A trigger type of `Manual` indicates that the pipeline was triggered manually.
@pipeline().TriggerId	ID of the trigger that invoked the pipeline
@pipeline().TriggerName	Name of the trigger that invoked the pipeline
@pipeline().TriggerTime	Time of the trigger run that invoked the pipeline. This is the time at which the trigger **actually** fired to invoke the pipeline run, and it may differ slightly from the trigger's scheduled time.
@pipeline().GroupId	ID of the group to which pipeline run belongs.
@pipeline()?.TriggeredByPipelineName	Name of the pipeline that triggers the pipeline run. Applicable when the pipeline run is triggered by an ExecutePipeline activity. Evaluate to *Null* when used in other circumstances. Note the question mark after @pipeline()
@pipeline()?.TriggeredByPipelineRunId	Run ID of the pipeline that triggers the pipeline run. Applicable when the pipeline run is triggered by an ExecutePipeline activity. Evaluate to *Null* when used in other circumstances. Note the question mark after @pipeline()

- **Schedule Trigger Scope**

Variable Name	Description
@trigger().scheduledTime	Time at which the trigger was scheduled to invoke the pipeline run.
@trigger().startTime	Time at which the trigger **actually** fired to invoke the pipeline run. This may differ slightly from the trigger's scheduled time.

- **Tumbling Window Trigger Scope**

Variable Name	Description
@trigger().outputs.windowStartTime	Start of the window associated with the trigger run.
@trigger().outputs.windowEndTime	End of the window associated with the trigger run.
@trigger().scheduledTime	Time at which the trigger was scheduled to invoke the pipeline run.

Variable Name	Description
@trigger().startTime	Time at which the trigger **actually** fired to invoke the pipeline run. This may differ slightly from the trigger's scheduled time.

- **Storage event trigger scope**

Variable Name	Description
@triggerBody().fileName	Name of the file whose creation or deletion caused the trigger to fire.
@triggerBody().folderPath	Path to the folder that contains the file specified by `@triggerBody().fileName`. The first segment of the folder path is the name of the Azure Blob Storage container.
@trigger().startTime	Time at which the trigger fired to invoke the pipeline run.

- **Custom event trigger scope**

@triggerBody().event.eventType	Type of events that triggered the Custom Event Trigger run. Event type is customer-defined field and take on any values of string type.
@triggerBody().event.subject	Subject of the custom event that caused the trigger to fire.
@triggerBody().event.data._keyName_	Data field in custom event is a free from JSON blob, which customer can use to send messages and data. Please use data.*keyName* to reference each field. For example, @triggerBody().event.data.callback returns the value for the *callback* field stored under *data*.
@trigger().startTime	Time at which the trigger fired to invoke the pipeline run.

21. Supported File Formats in Azure Data Factory

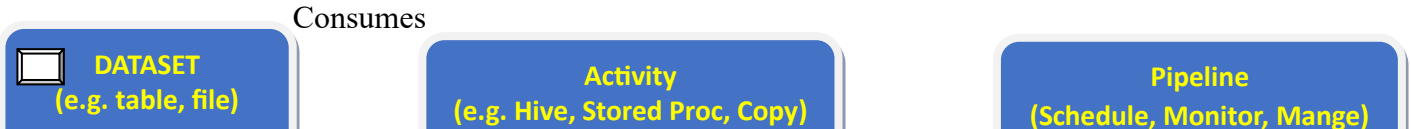
- AVRO format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

22. How to Create an Azure SSIS Integration Runtime in Azure Data Factory Step by Step

Azure Data Factory Activities and Its Types

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:



Produces

Is a Logical Grouping of

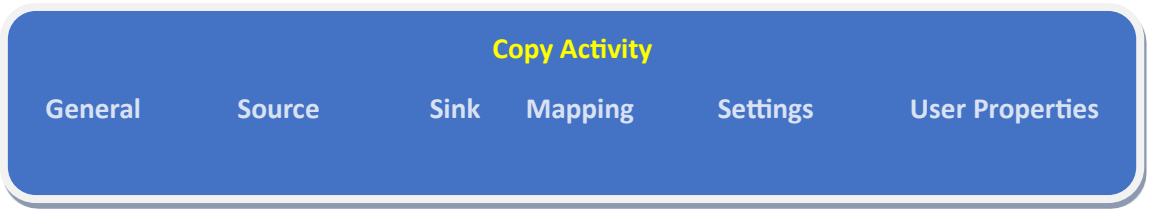
An Input dataset simply tells you about the input data and it's schema. And an Output dataset will tell you about the output data and its schema. You can attach zero or more Input datasets and one or more Output datasets. Activities in Azure Data Factory can be broadly categorized as:

1- Control Flow Activities.
2- Data Movement Activities.
3- Data Transformation Activities.

Control Flow Activity:

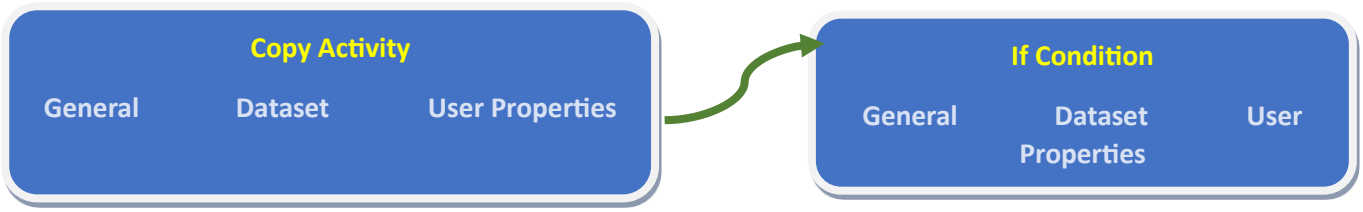
a. Copy Activity:

- Copies data from one data store to another.
- Can perform transformations on data
- Most popular cloud and on-premises data stores.
- Copy data activity is core activity in ADF. You can copy data from more than 90 connectors one to another.
- Connector Specific properties.



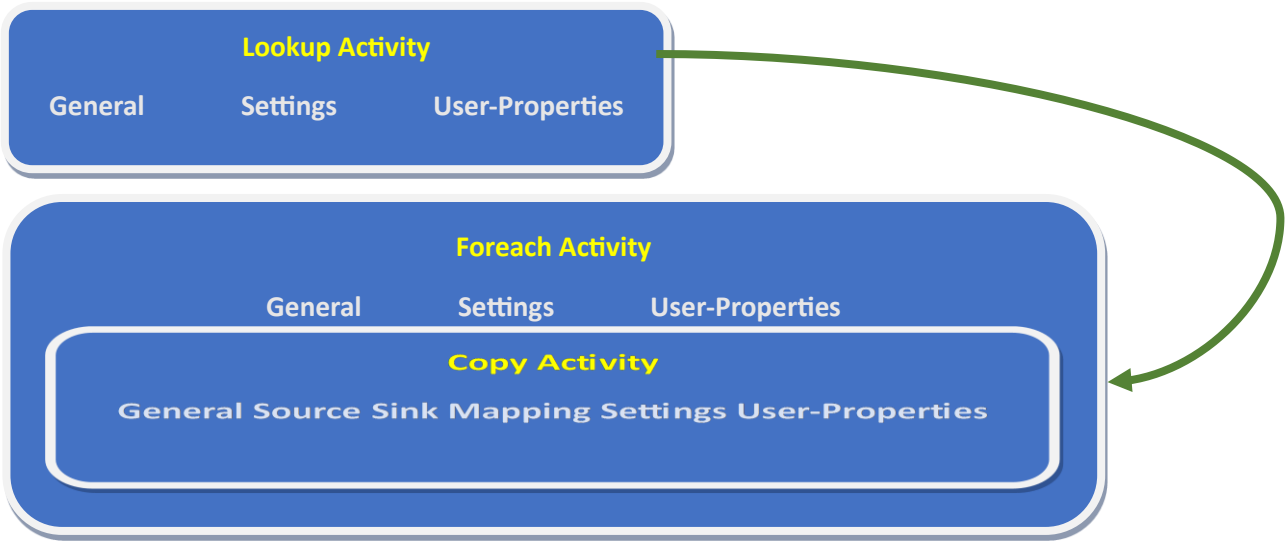
b. Get Metadata Activity:

- Activity helps in retrieving metadata information from a data store.
- Depends on the data store connector that is used.
- Maximum size of data that can be copied 4MB.
- Information related to metadata such as Child Items, Exists, Item Name, Item Type, and Last Modified.
- You can use the Get Metadata Activity to retrieve the Metadata of any data in Azure Data Factory.



c. Lookup Activity:

- Lookup Activity helps in retrieving lookup datasets from any Azure data factory-supported resources.
- From the database, we can get dataset using Table, Query, and Stored Procedure.
- Lookup Activity in ADF
- Lookup activity can retrieve a dataset from any of the Azure Data Factory- supported data sources.
- Lookup activity reads and returns the content of a configuration file or table. It also returns the result of executing a query or stored procedure.
- The output from Lookup activity can be used in a subsequent Activity.



d. Delete Activity:

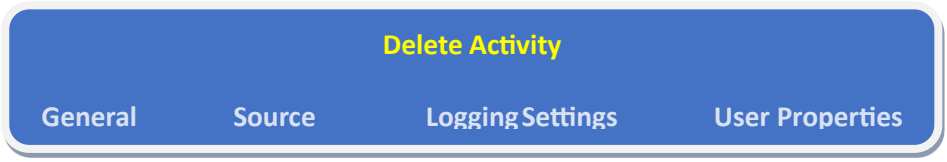
- You can use the Delete Activity in Azure Data Factory to delete files or folders from on-premises storage stores or cloud storage stores.
- Deleted files or folders cannot be restored.

Supported data stores

- Azure Blob storage
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure Files
- FTP
- Amazon S3
- Amazon S3 Compatible Storage
- Oracle Cloud Storage
- HDFS

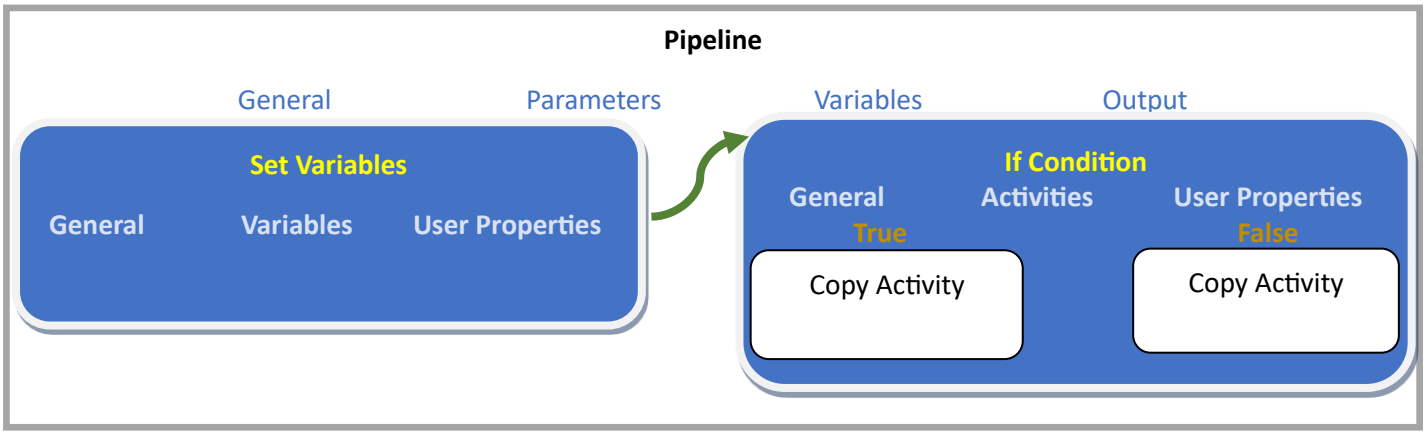
File system data stores

- File System
- SFTP
- Amazon S3
- Google Cloud Storage



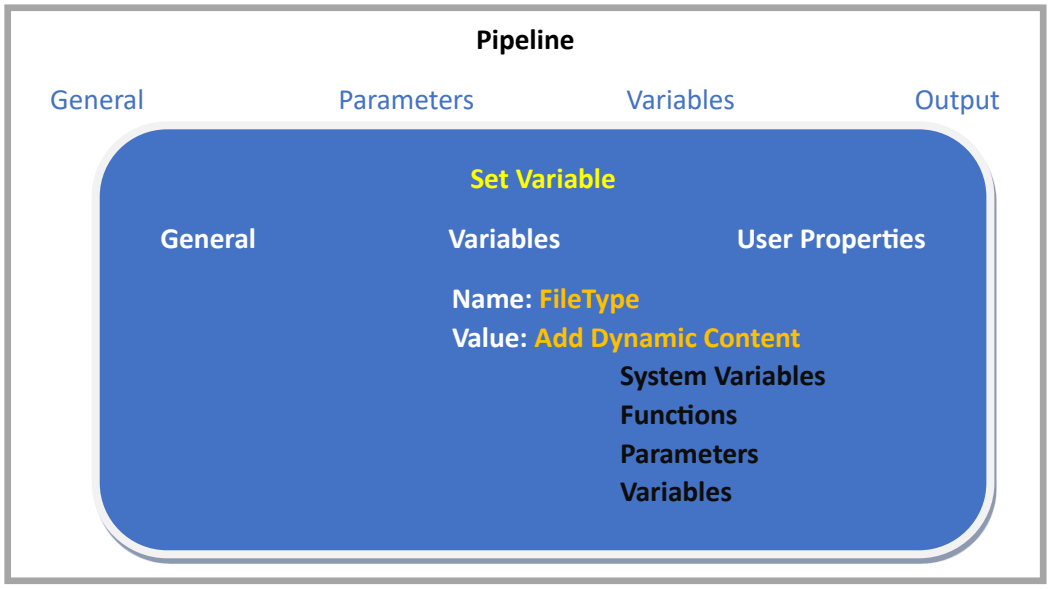
Variables in ADF:

- Variables are like internal to pipeline and they can be changed inside your pipeline
- Variables support 3 data types: string, bool, array
- We refer these user variables as below:
- @variables(‘variableName')
- Variable ---> set values for these variables inside pipelines--->set variable.



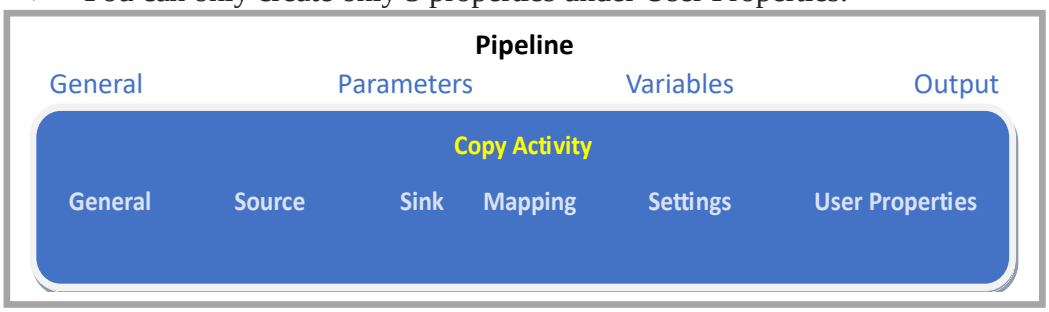
e. Set Variable Activity:

- Use the Set Variable activity to set the value of an existing variable of type String, Bool, or Array defined in a Data Factory pipeline.



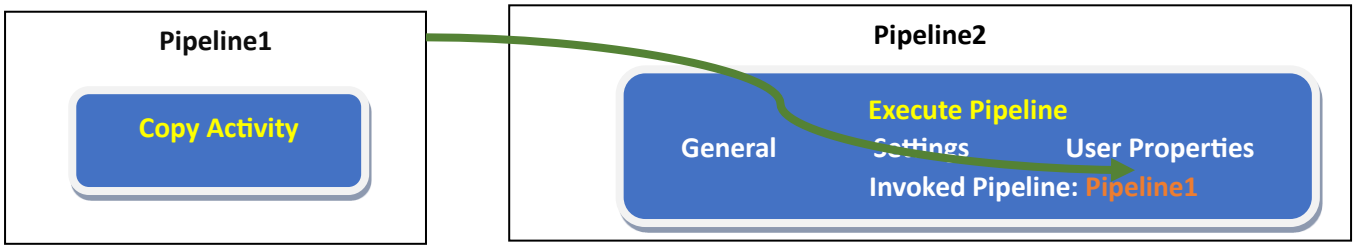
User Properties:

- User Properties helps to view addition information while monitoring Activity -runs
- You can only create only 5 properties under User Properties.



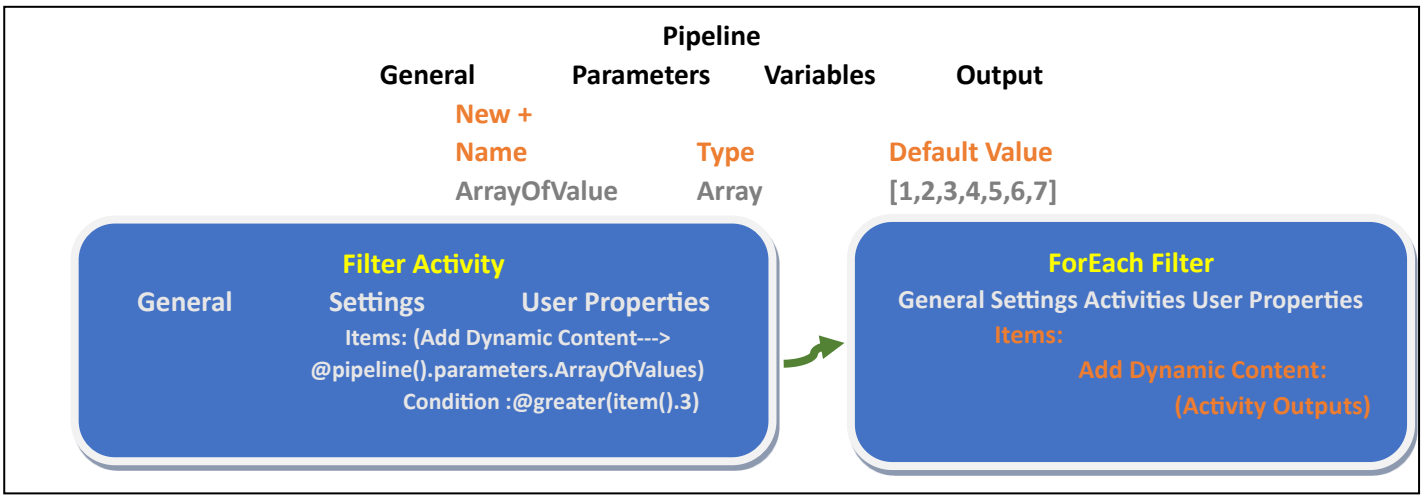
f. Execute Pipeline Activity:

- The Execute Pipeline activity allows a Data Factory pipeline to invoke another Pipeline.



g. Filter Activity:

- You can use a Filter activity in a pipeline to apply a filter expression to an input array.

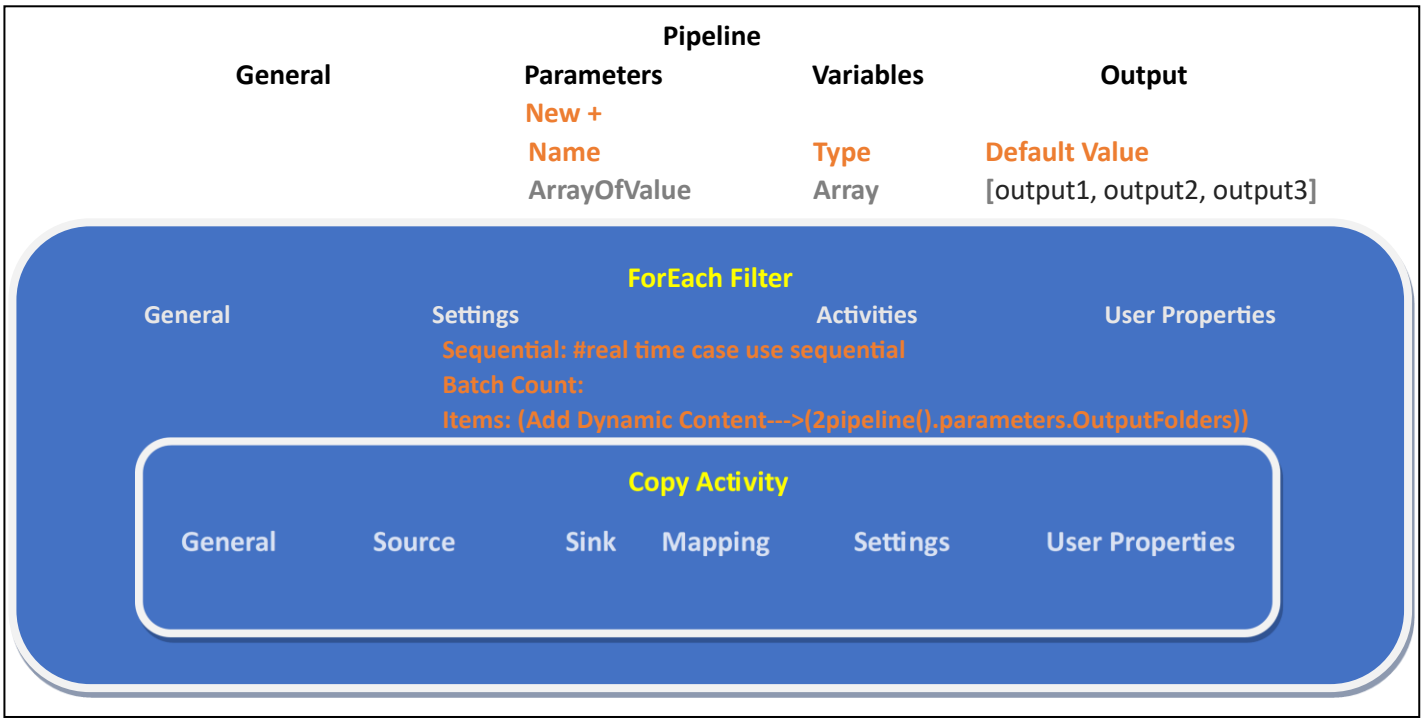


h. ForEach Activity:

- ForEach Activity defines a repeating control flow in your pipeline. This activity is used to iterate over a collection and executes specified activities in a loop.
- The items property is collection and each item inside collection is referred by @item ().

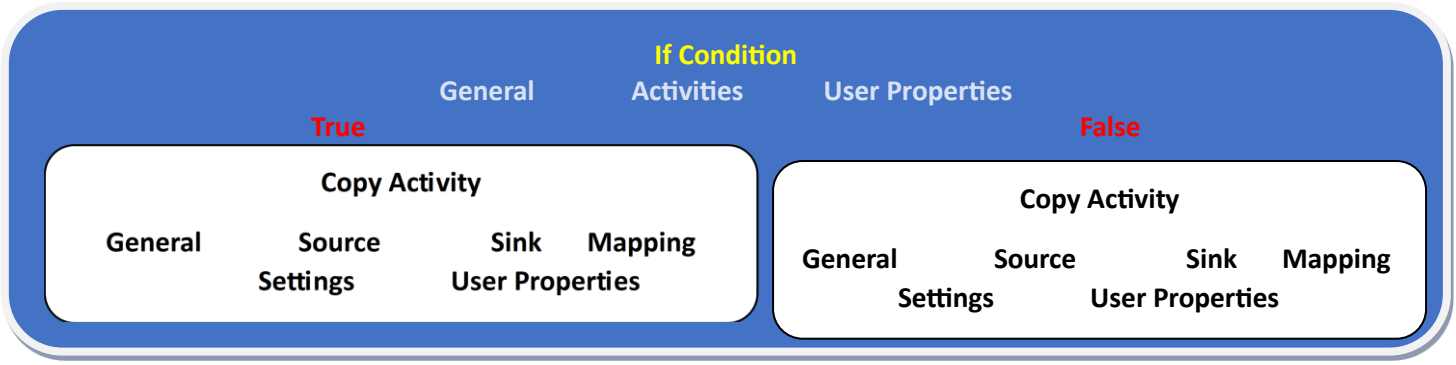
Example: if items is an array: [1, 2, 3], @item() returns 1 in the first iteration, 2 in the second iteration, and 3 in the third iteration.

adfdemo/input/ data. txt
adfdemo/output1
adfdemo/output2
adfdemo/output3



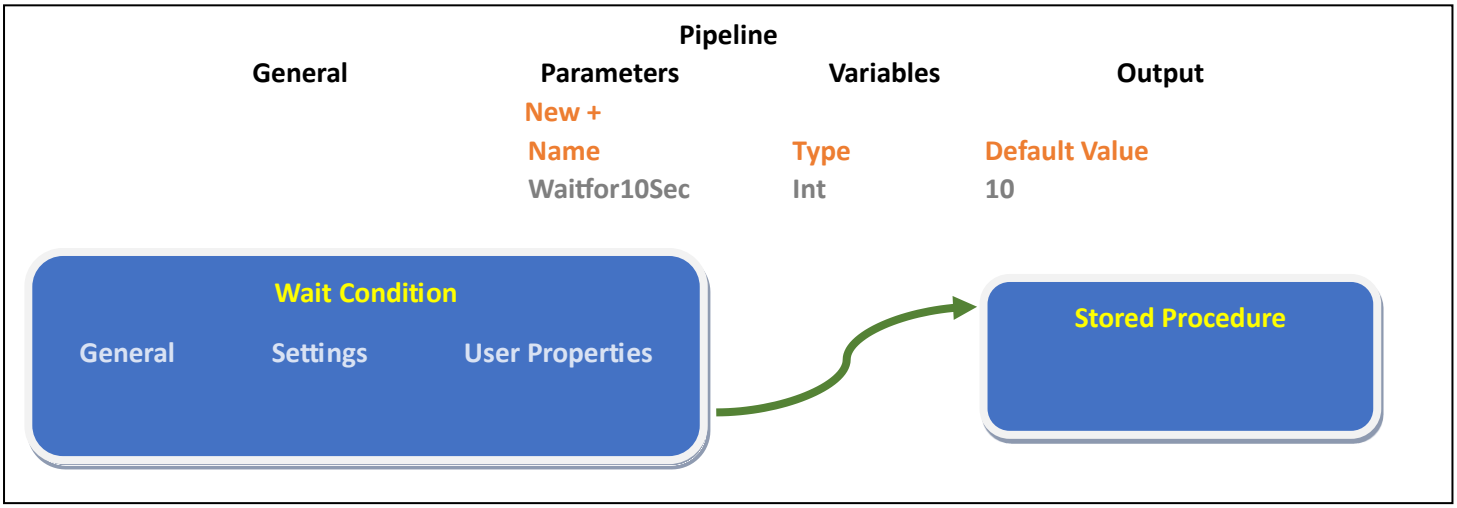
i. If-Condition Activity:

- If Condition activity provides the same functionality that an if statement provides in programming languages.
- If the expressions resolve to true, then set of activities will run and if the expression resolves to false then another set of activities will run.



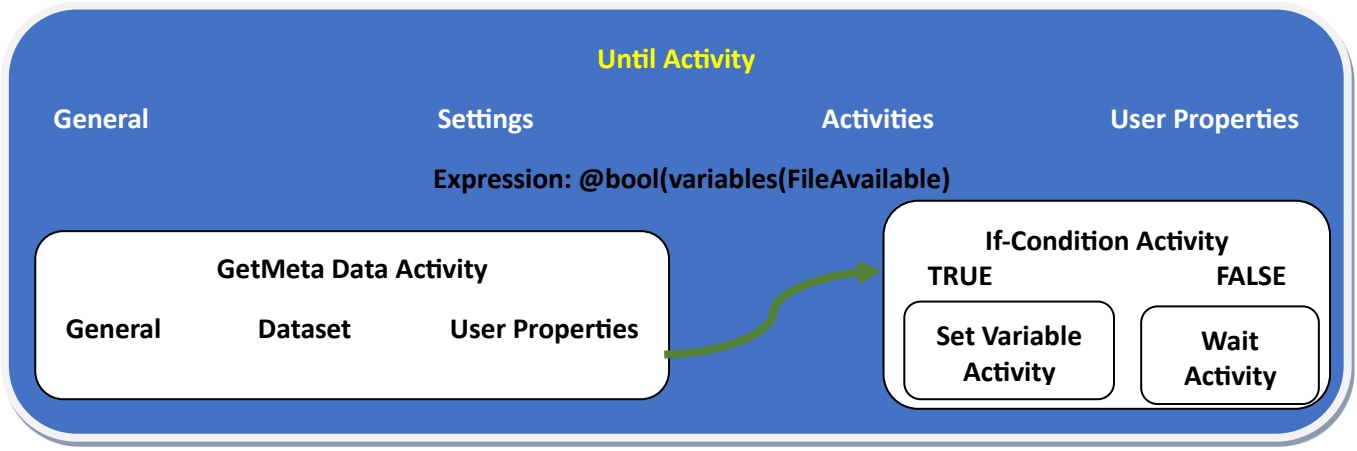
j. Wait Activity:

- When you use a Wait activity in a pipeline, the pipeline waits for the specified period of time before continuing with execution of subsequent activities.



k. **Until Activity:**

- This activity is like Do Until or Do While Activities in programming languages. That means, it is guarantee that at least one loop will defiantly run as condition evaluation will happen at the end of loop.
- It executes a set of activities in a loop until the condition associated with the activity evaluates to true.

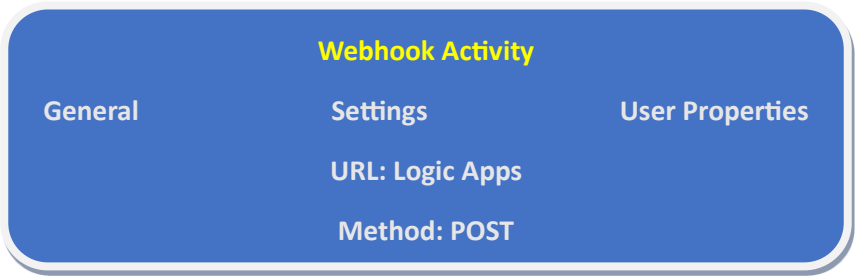


l. Web Activity:

- Web Activity can be used to call a custom REST endpoint from a Data Factory pipeline.
- You can pass Datasets and Linked Services also to REST API.
- Web Activity can call only publicly exposed URLs. It's not supported for URLs that are hosted in a private virtual network.
- Web is Asynchronous.

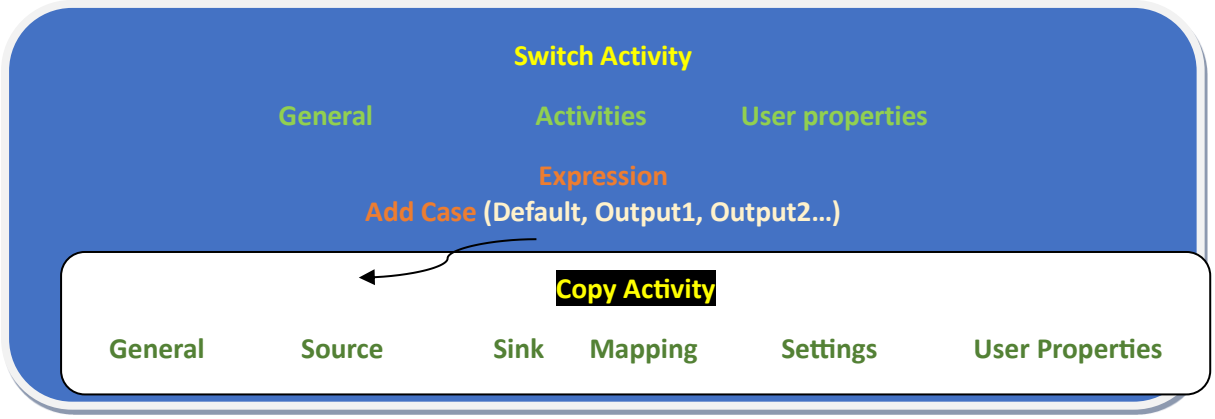
m. WebHook Activity:

- WebHook Activity we can call an endpoint and pass it a callback URL. The pipeline run waits for the callback invocation before it proceeds to the next activity.
- Because of this WebHook Activity in Synchronous in nature.
- When you issue REST call.
- ADF---> callback URL in request body.
- Webbook is Synchronous.



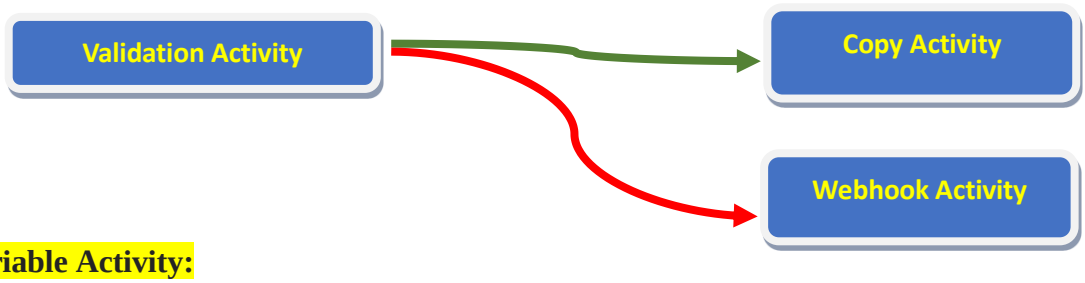
n. Switch Activity:

- Switch Activity provides the same functionality that Switch statement provides in Programming languages.
- It evaluates a set of activities corresponding to a case that matches the condition evaluation.



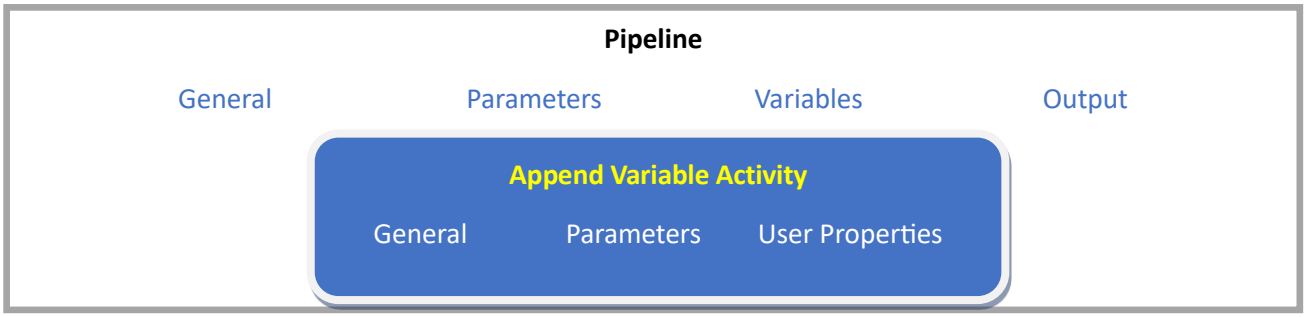
o. Validation Activity:

- You can use a Validation in a pipeline to ensure the pipeline only continues execution once it has validated the attached dataset reference exists.



p. Append Variable Activity:

- Use the Append Variable activity to add a value to an existing array variable defined in a Data Factory pipeline.
- Variables are internal to your pipelines.
- You can variables values you can change – set variable & append variable
- String, Bool, Array.



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.
2. **Monitor copy activity**
3. **Delete files using Delete activity**
4. **Copy data tool**
5. **Metadata driven copy data**

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 analyze 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.

Access Control:

Add User in Azure Active Directory

Create Users and Groups in Databricks

Grant Access: Manage, Edit, Run, Read

JDBC Connection Definition

```
jdbcHostname - "ss-demo-rajade. database.windows.net"
jdbcPort = 1433
jdbcDatabase "asqtdemo"
jdbcUsername = "rajade"
jdbcPassword = "Tester@234"
jdbcDriver = "com.microsoft.sqlserver.jdbc . SQLServerDriver"
jdbcUrl= f"jdbc :sqlserver: // {jdbcHostname} : {jdbcPort} ; databaseName = jdbcDatabase ; user = {j
dbcUsername} ; password={ jdbcPassword } "
```

Read from Azure SQL Database

empDF = spark.read.format(“ jdbc”.option(“url”, jdbcUrl).option(“dbtable” , ”dbo.emp”).load()
display(empDF)

goto user--->right side of window top click

add user in workspace--->if add any user into resource that user is part of Azure Active Directory.

Admin Console

Users Groups Global init scripts Workspace settings Notification Destinations

Create Group--->name(Devteam)

Members Entitlements Parent Groups

Add Users, groups or service principals

Goto Workspace:

Shared

Users--->select user right click---> Create
Clone
Import
Export
Permissions--->NAME PERMISSION
admins Can Manage inherited
rganesh0203@outlook.com Can manage
Select user. group or service principal. Can Read
Cancel
Copy Link Address

What is Secret Scope?

Secret is collection of secrets which is needed for Databricks development.

It prevents exposing sensitive details such as password, server name etc.,

Types of Secret Scopes:

Azure Key Vault-backed scopes.
Databricks-backed scopes.

URL:

- https://<databricks-instance>#secrets/createScope.
- Secret Name.
- Azure Key Vault DNS Name.
- Azure Key Vault Resource ID.

Create Mount Point to Integrate with ADLS

```
dbutils.fs.mount(
source =wasbs://demo@adlsrajademo.blob.core.windows. net",
mount _ point =”/mnt/adls_demo”,
extra_configs =
(“fs.azure.account.key.Adlsrajademo.blob.core.Windows.Net”:”
xQiWpRA5zn/RJKqD8AmkHv9pXVX33jFgxWxl_NAtcujCLdwh9ajIjwcAmCeGEFS
UEd.AStbvk0hA-="})
```

Write DataFrame into ADLS

```
from datetime import datetime
outpath = ‘/mnt/ad1s_demo/empOutputl_' +datetime.now() .strftime (“%Y%m%d%H%M%S”)
+’/’
empDF.write.format(“csv”).option(“header”, True).option(“sept” , “|”).save(outpath)
```

Databricks Workflows: Job Scheduling

Workflow page:

Jobs--->create jobs--->Task--->name--->Type (Notebook, Python script, Python Wheel SQL New, Delta Live Tables pipeline, dbt, JAR, Spark Submit.

Source--->(Workspace, Git Provider)
Path
Cluster (job cluster, all job purpose)
Add dependent libraries
Email Notifications
Create task
Second task
Third task
Create schedule

Job runs
Delta Live Tables

What is ADF:

- Orchestration tool
- Build ETL pipelines
- Schedule Pipelines

ADF Trigger vs Databricks Workflow

- ADF: Integrate many services

- Databricks: It is only for Databricks Notebooks

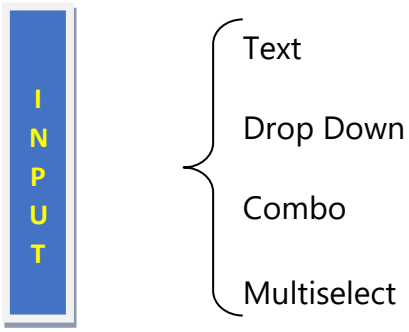
Components:

- ADB Notebook
- Output Parameters
- ADF linked services
- Input Parameters
- Cluster Selection

ADF:

Create pipeline--->add notebook (ADB)--->add parameters
Goto Azure databricks--->user setting--->generate access key.

What is Widget in Databricks?



Widgets in databricks enable users to add parameters to dashboards and notebooks. The Widgets in databricks are best applied while building a notebook or a dashboard that is re-executed previously with the different parameters. The Widgets are also used for exploring results quickly of the single query with other parameters. The Widgets are basically of four types are Text, Dropdown, Combobox, and Multiselect. The Text widget inputs a value in the text box in databricks. The Dropdown widget selects a value from the list of provided values in databricks. The Combobox widget is the combination of text and dropdown, and it sets a value from the provided list or inputs one in the text box. The Multiselect widget selects one or more values from the list of provided values.

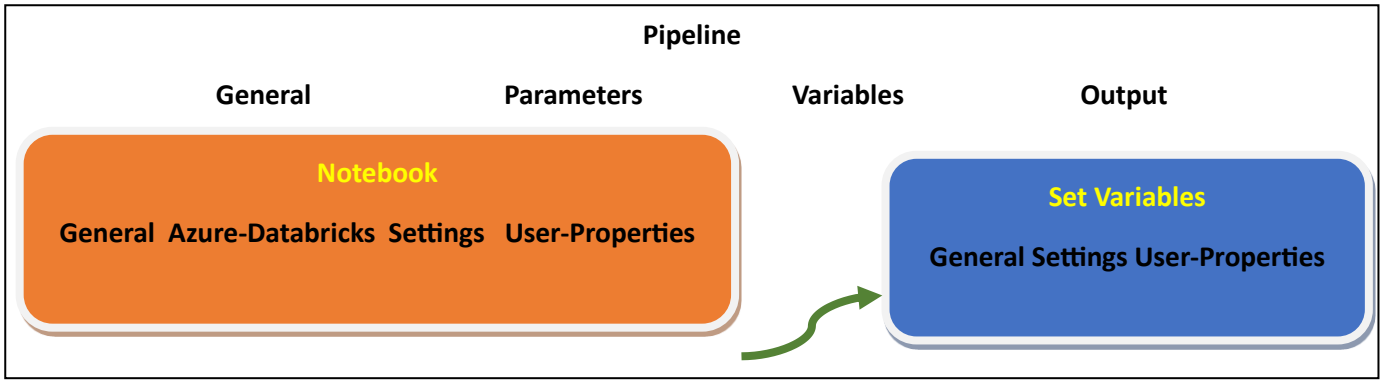
Databricks | Pyspark | Notebook Activity in Azure Data Factory with Output Parameter

Databricks Output Parameter:

dbutils.notebook.exit(<value>)

ADF Collect Output Value

@activity('ActivityName').output.runOutput



#Databricks Utilities

1. List Mounts
2. Read Files
3. Create Directories and Files
4. Widgets
5. List available utilities

   To list available utilities along with a short description for each utility, run `dbutils.help()` for Python or Scala.

   This example lists available commands for the Databricks Utilities.

   Python

   dbutils.help()

6. List available commands for a utility

   To list available commands for a utility along with a short description of each command, run `.help()` after the programmatic name for the utility.

   This example lists available commands for the Databricks File System (DBFS) utility.

   Python

   ```
   dbutils.fs.help()
   ```

7. Display help for a command

   To display help for a command, run `.help("<command-name>")` after the command name.

   This example displays help for the DBFS copy command.

   Python

   ```
   dbutils.fs.help("cp")
   ```

r)