Microsoft Certified

Azure Data Engineer Associate

💻 DP-203 Notes by Neil Bagchi

Topics to Learn (Microsoft Learn)

1.	**Basics (refresher)**	Notes link
2.	Azure Data Lake Storage	Notes link
3.	Data Factory	Notes link
4.	Azure Databricks	Notes link
5.	Azure Synapse Analytics	Notes link
6.	Azure Stream Analytics	Notes link
7.	Event Hubs	Notes link
8.	Azure Monitor	Notes link

Sites/ Courses followed:

1. Microsoft Learn Path- Link
2. Udemy Course by Eshant Garg (link) and Alan Rodrigues (link)
3. Practice Labs from Microsoft- Link

✅ Basics

1. **Data Types**
2. **Avro vs ORC vs Parquet**
3. **Availability Zones**
4. **OLTP vs OLAP**
5. **Data Lake vs Data Warehouse**

6. **Big Data Architecture**
7. **Partitioning Strategies**

✅ Data Types

Data is a collection of facts such as numbers, descriptions, and observations used to record information. We can classify data as *structured*, *semi-structured*, or *unstructured*.

Structured

Data that adheres to a fixed ***schema***, i.e. all of the data has the same fields or properties. This means the data structure is designed before any information is loaded into the system. Eg. Tabular data, CSV, spreadsheets

Semi-Structured

Data that has some structure but allows for some variation i.e doesn't fit neatly into tables such as NoSQL, JSON, XML, YAML etc

Unstructured

Data that doesn't have a specific structure such as documents, images, audio, video data, log data, and binary files.

✅ Optimized file formats for Storage

While human-readable formats for structured and semi-structured data can be useful, they're typically not optimized for storage space or processing. Some common optimized file formats include *Avro*, *ORC*, and *Parquet*:

Avro Row-based

> • *Writing new records is easy (efficient)*
> • *Reading parts of the records will involve reading the entire record thus being more memory intensive. (not efficient)*

Avro format works well with a message bus such as **Event Hubs** or **Kafka** that writes multiple events/messages in succession. Also good for workloads having a lot of ETL jobs, thus best for landing/raw zone.

ORC (Optimized Row Columnar format)

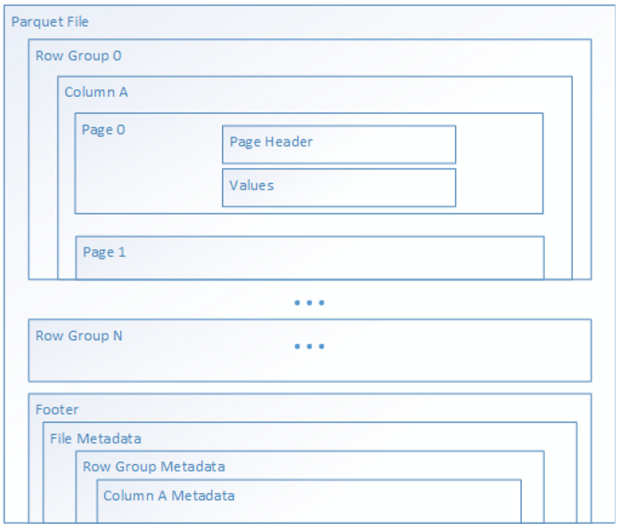
> • *Writes are not efficient*
> • *Reads are efficient*
> • *Highly efficient in terms of storage.*

It was developed for optimizing read and write operations in **Apache Hive**.

Parquet
Column based

> • *Writes are not efficient*
> • *Reads are efficient*
> • *Highly efficient in terms of storage but not as good as ORC*

Apache Parquet is an open-source file format that is optimized for read-heavy analytics pipelines. The columnar storage structure of Parquet **lets you skip over non-relevant data** making your queries much more efficient. This ability to skip also results in sending relevant data from storage to the analytics engine resulting in lower costs along with better performance. In addition, since similar data types (for a column) are stored together, Parquet lends itself friendly to efficient data compression and encoding schemes lowering your data storage costs as well.

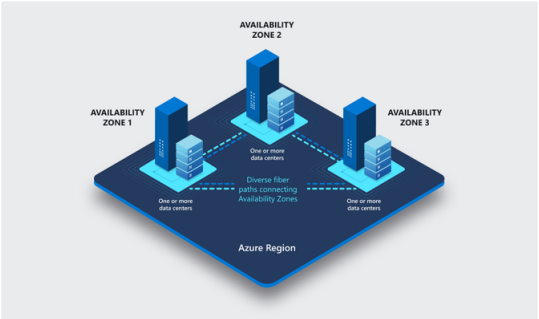


Services such as **Azure Synapse Analytics, Azure Databricks, and Azure Data Factory** have native functionality that takes advantage of Parquet file formats.

> 📌 ***TIP:*** *If you still need to store the data in any of the semi-structured formats such as CSV, JSON, XML, and so on, consider compressing them using* ***Snappy compression****.*

✅ Availability Zones

> *Availability zones are physically separate data centers within an Azure region. Each availability zone is made up of one or more data centers equipped with independent power, cooling, and networking. An availability zone is set up to be an isolation boundary. If one zone goes down, the other continues working.*



There's a minimum of three availability zones within a single region if applicable. However, not all regions have availability zones.

Option	Redundancy	Discussion
Locally redundant storage (LRS)	Three synchronous copies in same data center	Least expensive and least availability
Zone-redundant storage (ZRS)	Three synchronous copies in three AZs in the primary region	
Geo-redundant storage (GRS)	LRS + Asynchronous copy to secondary region (three more copies using LRS)	
Geo-zone-redundant storage (GZRS)	ZRS + Asynchronous copy to secondary region (three more copies using LRS)	Most expensive and highest availability

RA-GRS and RA-GZRS provide data access across both the region pairs at the same time and thus are more costly whereas, in GRS and GZRS, access to the other region pair only happens when one of the regions fails

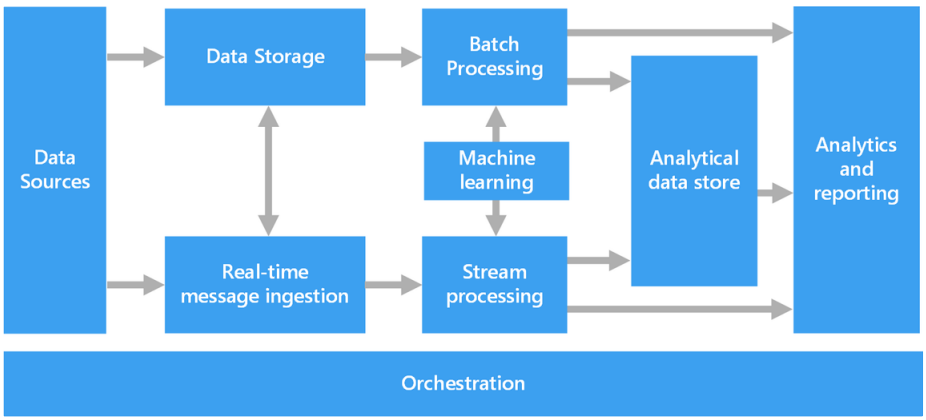
✅ OLTP vs OLAP

Transactional Processing (OLTP)	Analytical Processing (OLAP)
Analyses **individual entries**	Analyses **large batches** of data
Access to **recent** data	Access to **older** data going back years
Updates data frequently	Optimized for **reading operations**
Faster **real-time** access	**Long**-running jobs
Usually a **single** data source	**Multiple** data sources
MySQL, **Azure SQL Database**	Apache Hive, Teradata, **Azure Synapse Analytics**

✅ How is a data lake different from a data warehouse?

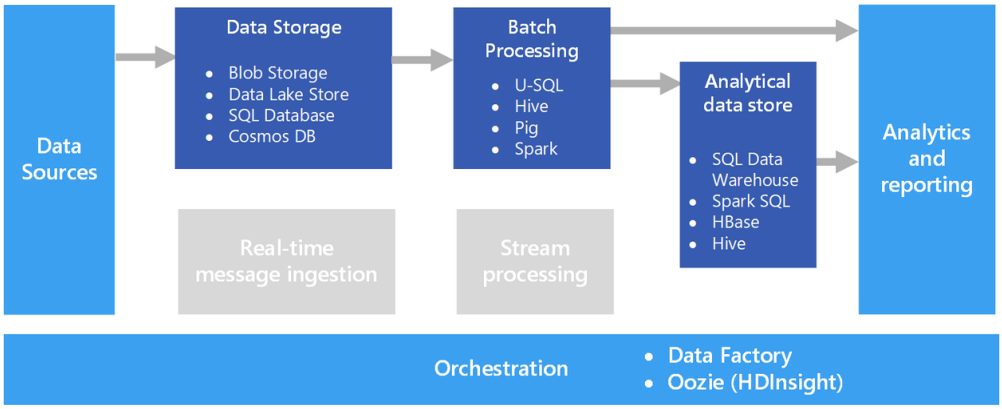
AREA	DATA LAKE	DATA WAREHOUSE
Data Store 10101 01010 00100	It can capture and retain unstructured, semi-structured, and structured data in its raw format. A Data Lake stores all types of data, irrespective of the source and structure.	It can capture and retain only structured data. A Data Warehouse stores data in quantitative metrics with their attributes. Data is transformed and cleansed.
Schema Definition	Typically, the schema is defined after data is stored. This offers high agility and data capture quite easily, but it requires work at the end of the process (schema-on-read).	Typically, a schema is defined prior to when data is stored. It requires work at the start of the process, but it offers performance, security, and integration (schema-on-write).
Data Quality	Any data that may or may not be curated (such a raw data).	Highly curated data that serves as the central version of the truth.
Users	A Data Lake is ideal for the users who indulge in deep analysis, like Data Scientists, Data Engineers, and Data Analysts.	A Data Warehouse is ideal for operational users like Business Analysts because of being well structured and easy to use and understand.
Price & Performance	The storage cost is relatively low, compared to a Data Warehouse, and querying results is better.	The storage cost is high, and querying results is time consuming.
Accessibility	A Data Lake has few constraints and is easily accessible. Data can be changed and updated quickly.	A Data Warehouse is structured by design, which makes it difficult to access and manipulate.

✅ Big Data Architecture (link)



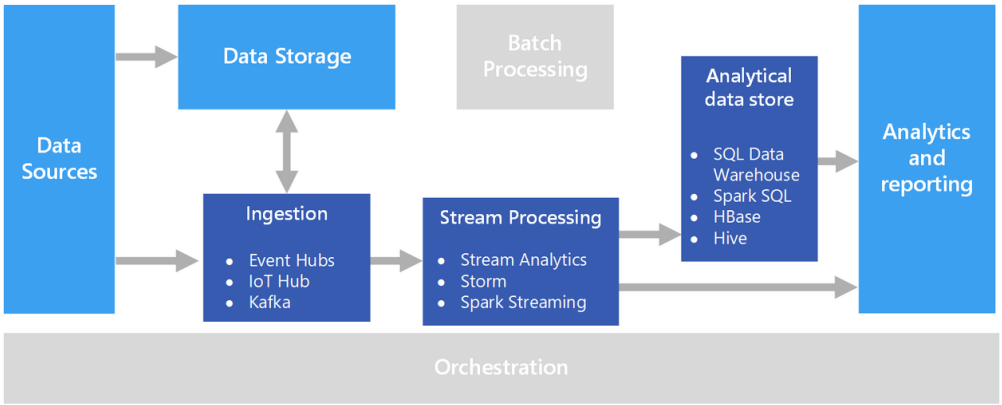
✅ Batch processing

Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually, these jobs involve reading source files, processing them, and writing the output to new files.



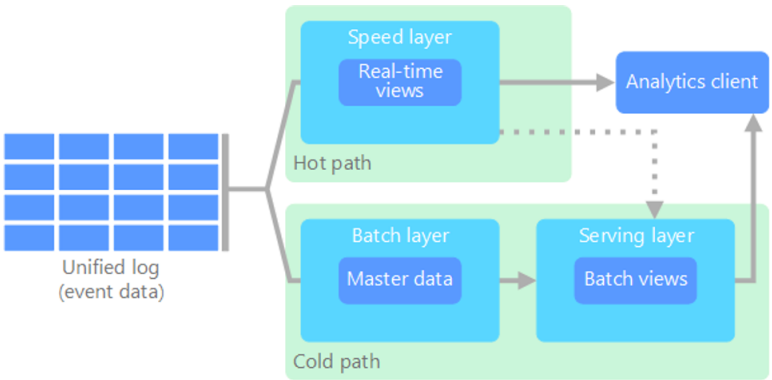
✅ Stream processing

After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink.



1) Lambda Architecture

One of the shortcomings of batch processing systems is the time it takes to process the data. One drawback of this approach is that it introduces latency, a batch pipeline might run for several hours - or sometimes even days - to generate the results.



The **lambda architecture** addresses this problem by using a combination of fast and slow pipelines. All data coming into the system goes through these two paths:

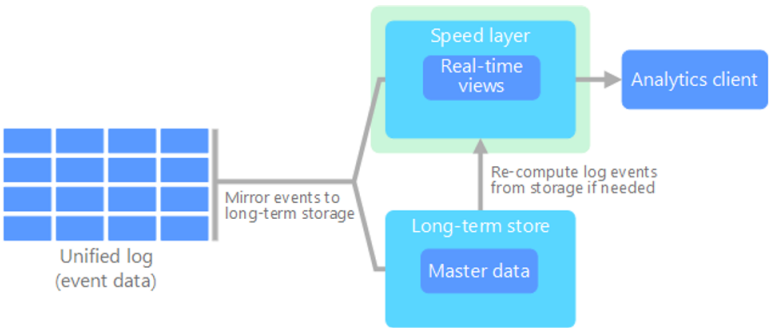
- A **batch layer** (cold/slow path) stores all of the incoming data in its raw form and performs batch processing on the data. The result of this processing is stored as a **batch view**.
- A **speed layer** (hot/fast path) analyzes data in real time. This layer is designed for low latency, at the expense of accuracy.

Both these pipelines feed into a **Serving layer** that updates the incremental updates from the fast path based on recent data into the baseline data from the slow path.

2) Kappa Architecture

A drawback to the lambda architecture is its **complexity**. Processing logic appears in two different places — the cold and hot paths — using different frameworks. This leads to duplicate computation logic and the complexity of managing the architecture for both paths.

In Kappa though, **all data flows through a single path, using a stream processing system.**



In Kappa architecture, the input component is a message queue such as an Apache Kafka or Azure Event Hubs queue, and all the processing is usually done through Azure Stream Analytics or Spark. Kappa architecture can be used for applications such as real-time ML and applications where the baseline data doesn't change very often.

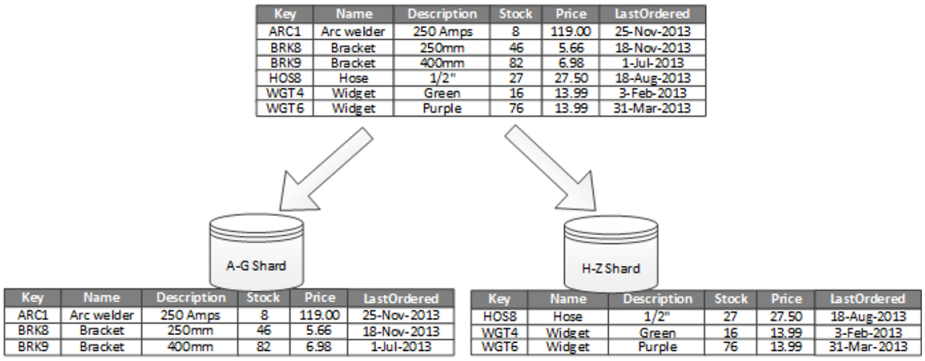
✅ Partitioning

In many large-scale solutions, data is divided into *partitions* that can be managed and accessed separately. Partitioning can improve scalability, reduce contention, and optimize performance. It can also provide a mechanism for dividing data by usage pattern.

There are three typical strategies for partitioning data:

1) Horizontal partitioning (often called *sharding*)

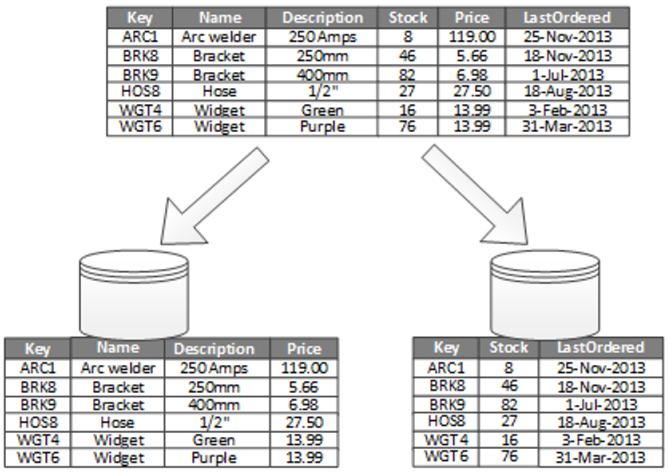
In this strategy, each partition is a separate data store, but all partitions have the same schema. Each partition is known as a *shard* and holds a specific subset of the data, such as all the orders for a specific set of customers.



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

2) Vertical partitioning

In this strategy, each partition holds a subset of the fields for items in the data store. The fields are divided according to their pattern of use. For example, frequently accessed fields might be placed in one vertical partition and less frequently accessed fields in another.



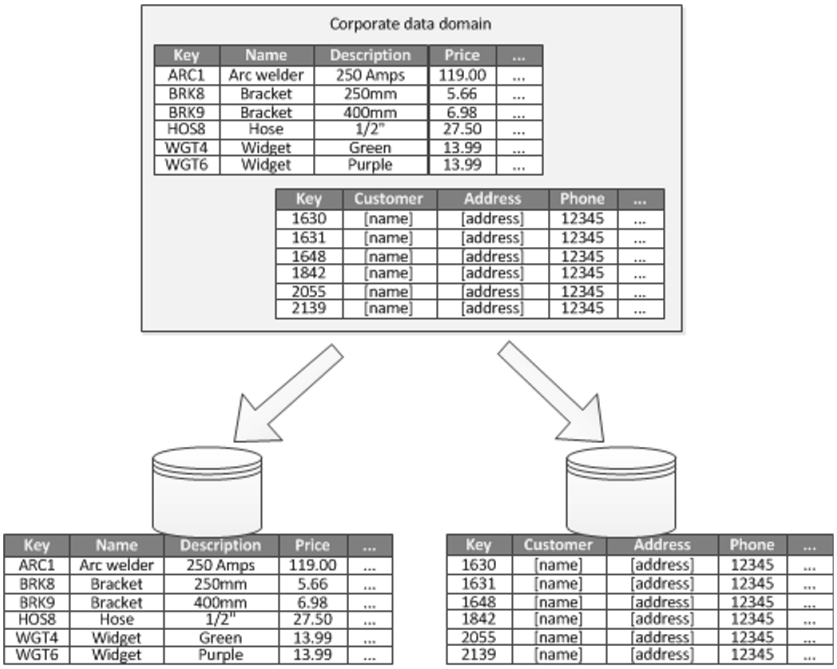
Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

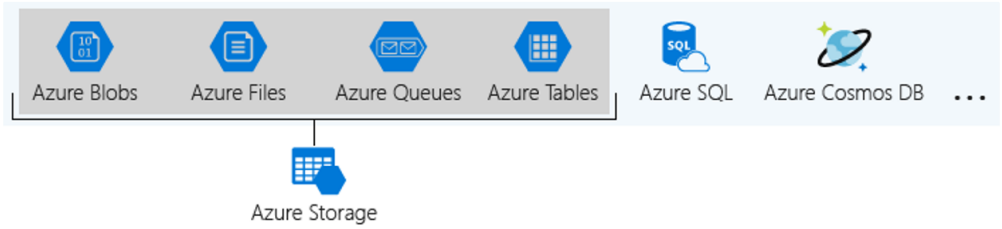
3) Functional partitioning

In this strategy, data is aggregated according to how it is used by each bounded context in the system. For example, an e-commerce system might store invoice data in one partition and product inventory data in another.

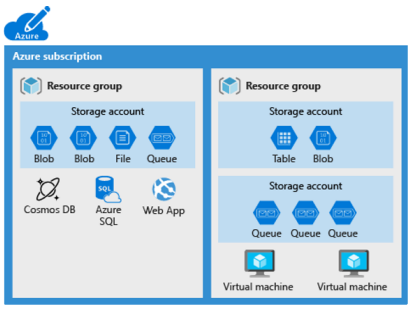


✅ Azure Storage

The following four data services together are called *Azure Storage*



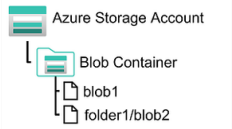
A **storage account** is a container that groups a set of Azure Storage services together. **Only** data services from Azure Storage can be included in a storage account (Azure Blobs, Azure Files, Azure Queues, and Azure Tables).



Other Azure data services, such as Azure SQL and Azure Cosmos DB, **are managed as independent Azure resources** and cannot be included in a storage account.

1) Azure Blob:

> ***Blob (binary large object) Storage is an object storage solution. It is the cheapest option to store unstructured data (no restriction on the type of data) that won't be queried.***



Every blob lives inside a **blob container**. You can store an unlimited number of blobs in a container and an unlimited number of containers in a storage account. Containers are "**flat**"; they can only store blobs, not other containers. **Blob Storage does not provide any mechanism for searching or sorting blobs by metadata.**

> 📌 *Technically, containers are "flat" and don't support any kind of nesting or hierarchy. But if you give your blobs hierarchical names that look like file paths (such as finance/budgets/2017/q1.xls), the API's listing operation can filter results to specific prefixes. This enables you to navigate the list as if it was a hierarchical system of files and folders. This feature is often called **virtual directories.***

Azure Blob Storage supports three different types of blob:

Block blobs:	Page blobs:	Append blobs:
Set of blocks of different sizes that can be uploaded independently and in parallel.	A page blob is **optimized to support random read and write operations.**	Specialized block blobs that support only appending new data (**no updating or deleting existing data**), but they're very efficient at it.

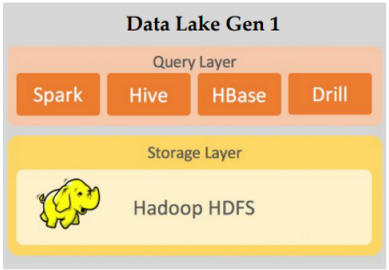
→ Hadoop HDFS vs DataLake (optional)

Hadoop consists of three core components –

- **Hadoop Distributed File System (HDFS)** – It is the storage layer of Hadoop.
- **Map-Reduce** – It is the data processing layer of Hadoop.
- **YARN** – It is the resource management layer of Hadoop.

Other than the core components of Hadoop, we have a bunch of ecosystem technologies. Some of the important ones are **Apache Spark, SQL Hive, Hbase, Sqoop, Pig, and Oozie**. All these together are called the Hadoop Ecosystem

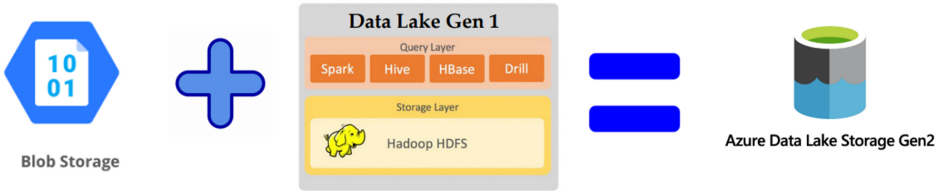
A data lake is an architecture within which Hadoop HDFS is just the storage component of that architecture. In a sense, both of these are complementary to each other. However, it is not necessary for a data lake to always use HDFS. Based on the requirements of the task, we can swap it with other technologies such as Apache Kafka for managing real-time data, NoSQL for transaction-oriented data, Hadoop for economical storage, or more recent Apache KUDU for large-scale analytics workloads.



Hadoop already has inbuilt advantages such as a fault-tolerant file system, the ability to run on commodity hardware, etc. Microsoft utilized these advantages by creating Data Lake Gen 1 which is basically Hadoop in the cloud.

However, with time, requirements evolved in terms of processing as well as storage capabilities

Here enters Microsoft Blob Storage which could store massive amounts of unstructured data. Blob storage is a general-purpose object storage that provides cheap storage. So Microsoft combined all the good features of Blob storage and DataLake Gen1 to create **Azure Data Lake Gen2**

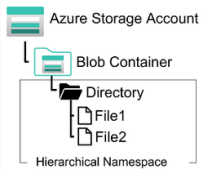


Differences:

Hadoop 2.0	ADLS Gen2
Clusters are tightly coupled with HDFS	Storage is separate from clusters
On stopping the cluster, all data is lost	We can stop the cluster without losing any data
Costly: Clusters have to keep on running even if there is no processing and pay for both storage and cluster	Cost efficient: Only pay for storage when processing is not required

1.a) Data Lake Storage Gen2 (optimized for big data analytics)

> ***Enhanced Blob Storage for enterprise big data analytics (hierarchical namespace). It provides low-cost, tiered storage, with high availability/ disaster recovery.***



• ABFS (Azure blob file system) is a dedicated driver for Hadoop running on Azure blob storage. Think of the data as if it's stored in a Hadoop Distributed File System (HDFS) which means that Azure Data Lake Storage organizes the stored data into a hierarchy of directories and subdirectories, much like a file system, for easier navigation. As a result, data processing requires fewer computational resources, reducing both the time and cost.

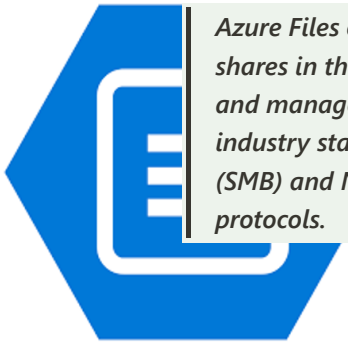
• Azure Data Lake Storage Gen2 implements an access control model that supports both Azure role-based access control (Azure RBAC) and POSIX-like access control lists (ACLs). You can set permissions at a directory level or file level for the data stored within the data lake. (more on this later)

Below is a common example we see for data that is structured by date:

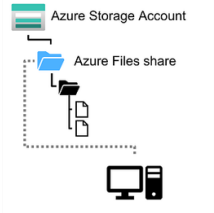
- `/DataSet/YYYY/MM/DD/datafile_YYYY_MM_DD`
- `{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/`

> 📌 ***TIP:*** *Avoid putting date folders at the beginning as it makes applying ACLs to every subfolder more tedious.*

2) Azure Files:



> *Azure Files offers fully managed file shares in the cloud that can be accessed and managed like a file server using the industry standard Server Message Block (SMB) and Network File System (NFS) protocols.*

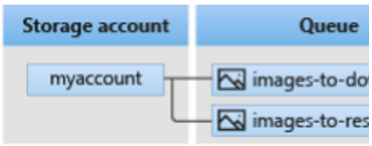


File shares can be used for many common scenarios:

- Shared data between on-premises applications and Azure VMs to allow migration of apps to the cloud over a period of time.
- Storing shared configuration files for VMs, tools, or utilities so that everyone is using the same version. Log files such as diagnostics, metrics, and crash dumps.

3) Azure Queue:

> *A messaging store used to store a large number of messages that can be accessed asynchronously between the source and the destination.*

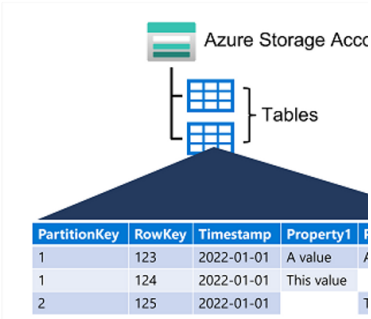


There are two types of queues: **Storage queues** and **Service Bus**.

- Storage queues can be used for simple asynchronous message processing. They can store up to 500 TB of data (per storage account) and each message can be up to 64 KB in size.
- Service Bus provides advanced features plus the message sizes can be up to 1 MB but the overall size is capped at 80 GB.

4) Azure Table:

> *A NoSQL store that hosts unstructured data independent of any schema. It makes use of tables containing key-value data items.*



It makes use of tables containing key-value data items but is **not similar to a relational database**. Thus there is no concept of relationships, stored procedures, secondary indexes, or foreign keys. Data is **denormalized**, with each row holding the entire data for a logical entity. To help ensure fast access, Azure Table Storage splits a table into partitions

---

✅ Data archiving solution

Hot access tier:

Higher storage costs, but lower access and transaction costs. Optimized for storing data that is accessed frequently (for example, images for your website).

Cool access tier:

Lower storage costs, but higher access and transaction costs. Optimized for data that is infrequently accessed and stored for **at least 30 days** (for example, invoices for your customers).

Archive access tier (*available only at individual blob level*):

Lowest storage costs, but highest access, and transaction costs. Appropriate for data that is rarely accessed and stored for **at least 180 days**, with flexible latency requirements (for example, long-term backups)

> 📌 *To read data in archive storage, you must first change the tier of the blob to hot or cool. This process is known as **rehydration** and can take hours to complete.*

✅ Data life cycle management

We can define policies such as how long a particular data needs to be in the Hot Access, when to move the data between the different access tiers, when to delete blobs, and so on. ***Azure runs data life cycle policies only once a day, so it could take up to 24 hours for your policies to kick in.***

✅ Optimizing data lake for scale and performance

- **Optimize for high throughput** – target getting at least a few MBs (higher the better) per transaction.

- **Optimize data access patterns** – reduce unnecessary scanning of files, read only the data you need to read

1) File sizes and number of files

Analytics engines (ingest or data processing pipelines) incur overhead for every file they read (related to the listing, checking access, and other metadata operations) and too many small files can negatively affect the performance of your overall job. Further, when you have files that are too small (in the KBs range), the amount of throughput you achieve with the I/O operations is also low, requiring more I/Os to get the data you want. In general, it's a best practice to **organize your data into larger-sized files (target at least 100 MB or more) for better performance**.

In a lot of cases, if your raw data (from various sources) itself is not large, you have the following options to ensure the data set your analytics engines operate on is still optimized with large file sizes.

- Add a data processing layer in your analytics pipeline to coalesce data from multiple small files into a large file. You can also use this opportunity to store data in a read-optimized format such as Parquet for downstream processing.
- In the case of processing real-time data, you can use a real-time streaming engine (such as Azure Stream Analytics or Spark Streaming) in conjunction with a message broker (such as Event Hub or Apache Kafka) to store your data as larger files.

2) Partitioning Strategy

An effective partitioning scheme for your data can improve the performance of your analytics pipeline and also reduce the overall transaction costs incurred with your query. In simplistic terms, partitioning is a way of organizing your data by grouping datasets with similar attributes together in a storage entity, such as a folder. When your data processing pipeline is querying for data with that similar attribute (E.g. all the data in the past 12 hours), the partitioning scheme (in this case, done by DateTime) lets you skip over the irrelevant data and only seek the data that you want.

For Blob Storage

If you remember, we discussed how every blob lives inside a **blob container** that we create. However, these containers are logical entities, so there is no guarantee that the data blobs we create will land in the same partition.

However, Azure implements **range partitioning** using **lexical sequence** i.e. filenames File1, File2, .. may end up in the same partition when compared to OldFile1, OldFile2, .. which may end up in a different partition.

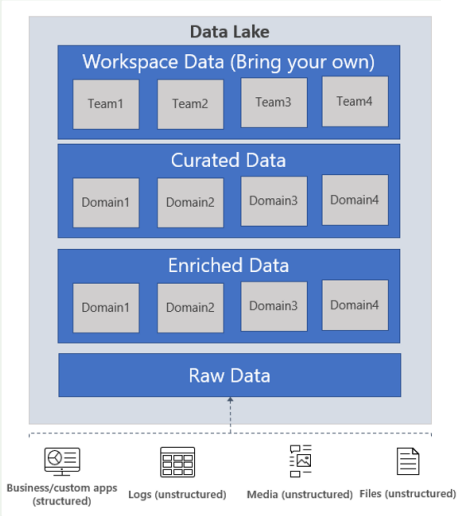
Azure Storage uses <account name + container name + blob name> as the partition key.

For ADLS Gen2

Since ADLS is hierarchical in nature, implementing a folder structure will take care of the partitioning strategy. Try to follow something like
`{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/`

✅ How to organize data? (Best Practice - more reading)

As an example, think of the raw data as a lake/pond with water in its natural state, the data is ingested and stored as is without transformations, and the enriched data is water in a reservoir that is cleaned and stored in a predictable state (schematized in the case of our data), the curated data is like bottled water that is ready for consumption. Workspace data is like a laboratory where scientists can bring their own for testing. It's worth noting that while all these data layers are present in a single logical data lake, they could be spread across different physical storage accounts. In these cases, having a metastore is helpful for discovery.



Raw data: This is data as it comes from the source systems. This data is stored as is in the data lake and is consumed by an analytics engine such as Spark to perform cleansing and enrichment operations to generate the curated data. The data in the raw zone is sometimes also stored as an aggregated data set, e.g. in the case of streaming scenarios, data is ingested via a message bus such as Event Hub, and then aggregated via a real-time processing engine such as Azure Stream Analytics or Spark Streaming before storing in the data lake.

Enriched data: This layer of data is the version where raw data (as is or aggregated) has a defined schema and also, and the data is cleansed, and enriched (with other sources), and is available to analytics engines to extract high-value data.

Curated data: This layer of data contains the high-value information that is served to the consumers of the data – the BI analysts and the data scientists. This data has structure and can be served to the consumers either as is (E.g. data science notebooks) or through a data warehouse. Data assets in this layer are usually highly governed and well documented.

Workspace data: In addition to the data that is ingested by the data engineering team from the source, the consumers of the data can also choose to bring other data sets that could be valuable. In this case, the data platform can allocate a workspace for these consumers so they can use the curated data along with the other data sets they bring to generate valuable insights.

Archive data: This is an organization's data 'vault' - that has data stored to primarily comply with retention policies and has very restrictive usage, such as supporting audits. You can use the Cool and Archive tiers in ADLS Gen2 to store this data.

✅ Latency metrics

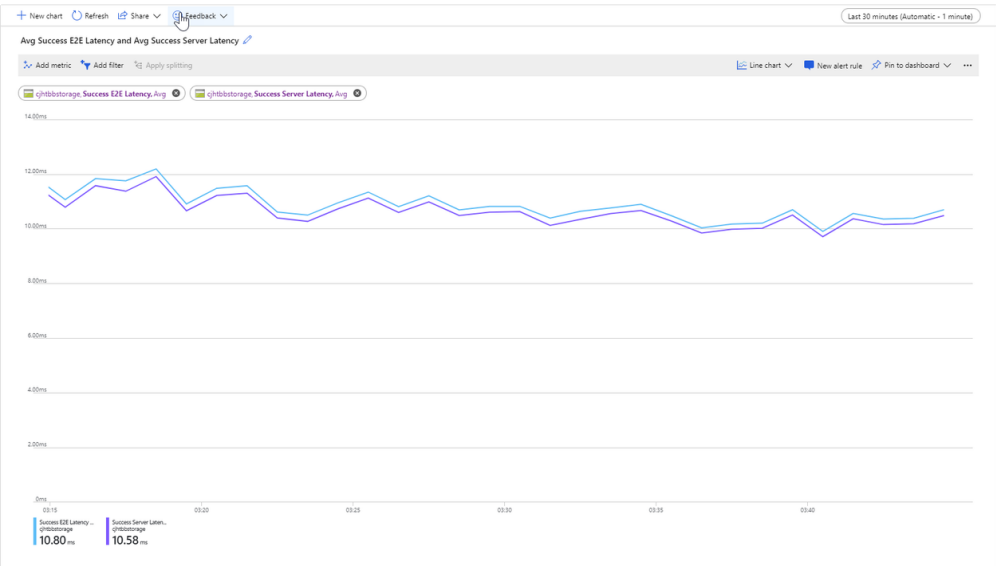
Request rate is measured in Input/output operations per second (IOPS). The request rate is calculated by dividing the time it takes to complete one request by the number of requests per second. E.g. Let us assume that a request from a single thread application with one outstanding read/write operation takes 10 ms to complete.

Request Rate = 1sec/10ms = 1000ms/10ms = 100 IOPS

This means the outstanding read/write would achieve a request rate of 100 IOPS.

Azure Storage provides two latency metrics for block blobs. These metrics can be viewed in the Azure portal:

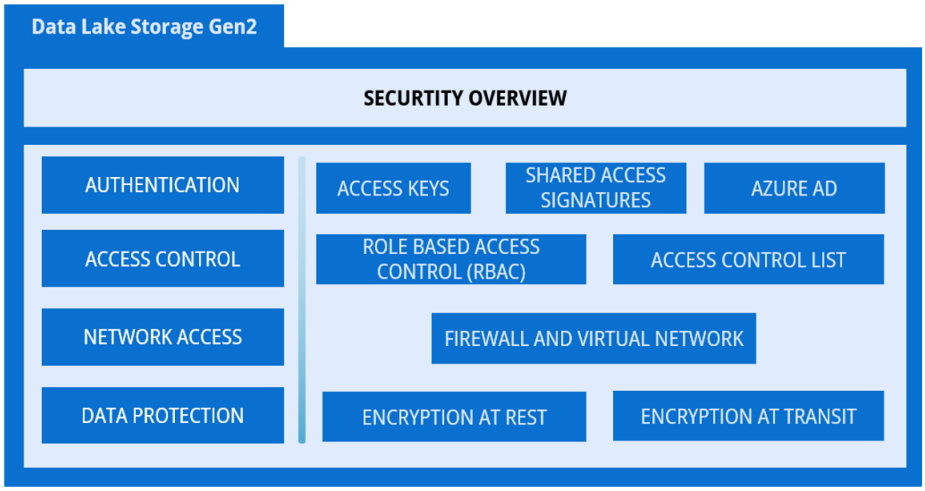
- **End-to-end (E2E) latency** measures the interval from when Azure Storage receives the first packet of the request from a client until Azure Storage receives a client acknowledgment on the last packet of the response.
- **Server latency** measures the interval from when Azure Storage receives the last packet of the request from a client until the first packet of the response is returned from Azure Storage.



✅ Storage Account Security Features

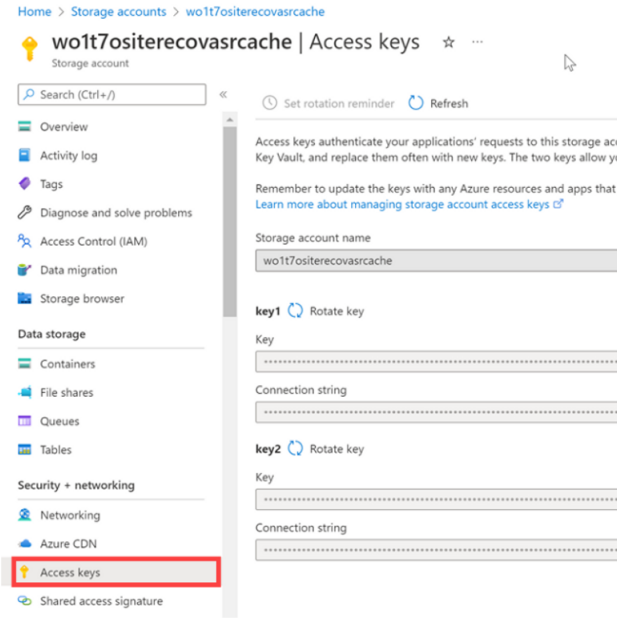
Azure Storage provides a layered security model. We can use this model to secure our storage accounts to a specific set of supported networks. Network rules allow only applications that request data over the specified networks to access our storage account.

Authorization is supported by a public preview of Azure Active Directory credentials (for blobs and queues), a valid account access key, or a shared access signature (SAS) token. Data encryption is enabled by default, and you can proactively monitor systems by using Advanced Threat Protection.



1) Access keys

Each storage account has **two unique *access keys*** that are used to secure the storage account. If your app needs to connect to multiple storage accounts, your app will require an access key for each storage account.

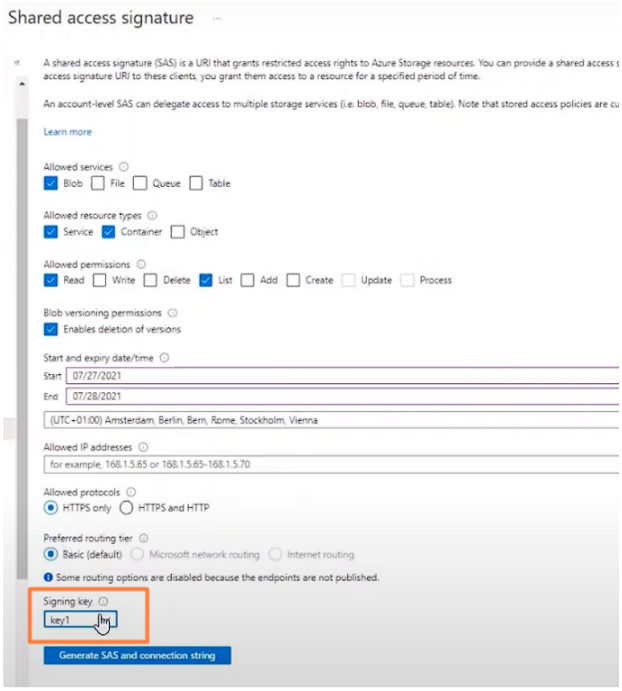


> 📌 ***TIP: Periodically rotate access keys*** *to ensure they remain private, just like changing your passwords. We can also use an* ***Azure Key Vault*** *to store the access key which includes the support to synchronize directly to the Storage Account and automatically rotate the keys periodically.*

2) Shared Access Signatures (SAS)

For external third-party applications, use a *shared access signature* (SAS). A SAS is a string that **contains a security token that can be attached to a URL.** You can use SAS to delegate access to storage objects and **specify constraints**, such as the **permissions** and the **time range of access**.

Azure doesn't track SAS after creation. Additionally, SAS tokens are tied to the access keys indirectly so to invalidate a SAS token, we need to regenerate/refresh the access keys. This can be painful to keep track of and continuously regenerate the keys. So an alternative is to use the Stored Access Policy.



📌 *A **stored access policy** groups together shared access signatures and provides additional restrictions for signatures that are bound by the policy. We can use a stored access policy to change the start time, expiry time, or permissions for a signature. We can also use a stored access policy to revoke a signature after it has been issued.*

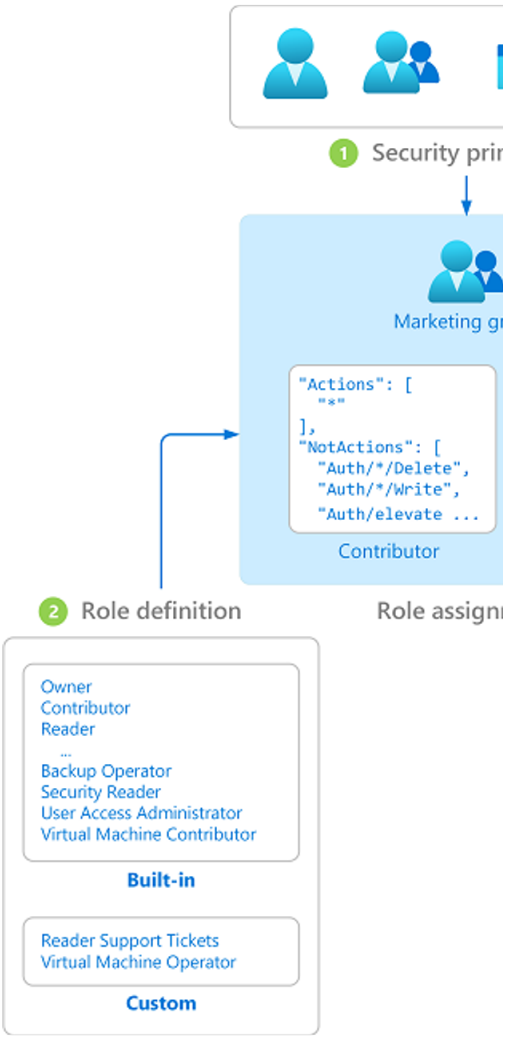
3) Role-based access control (RBAC)

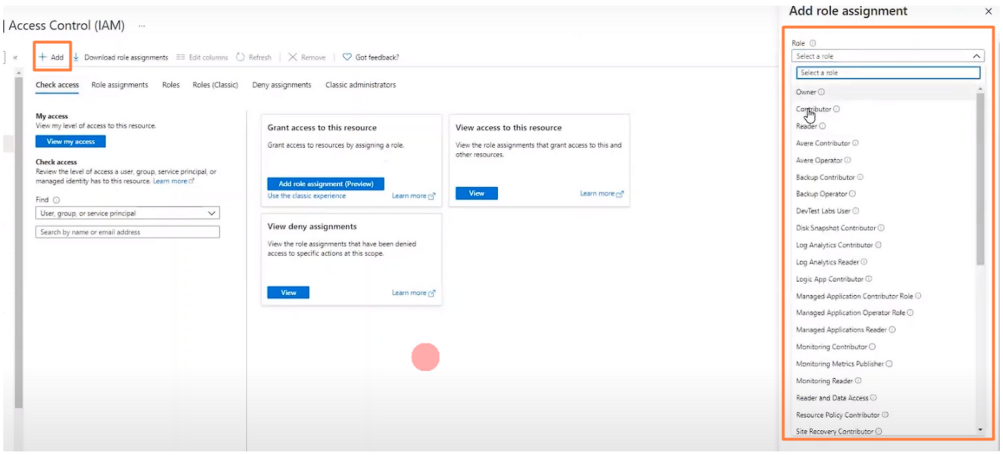
Azure role-based access control (Azure RBAC) helps manage who has access to Azure resources, what they can do with those resources, and what data they have access to. For ex:

1) **Security Pricipal:** A security principal is an object that represents a user, group, service principal, or managed identity that is requesting access to Azure resources. You can assign a role to any of these security principals.

2) **Role Definition:** A role definition is a collection of permissions. It's typically just called a role. A role definition lists the actions that can be performed, such as read, write, and delete. Azure has a huge list of predefined roles, such as Owner, Contributor, and Reader, etc with the right list of permissions, already assigned.

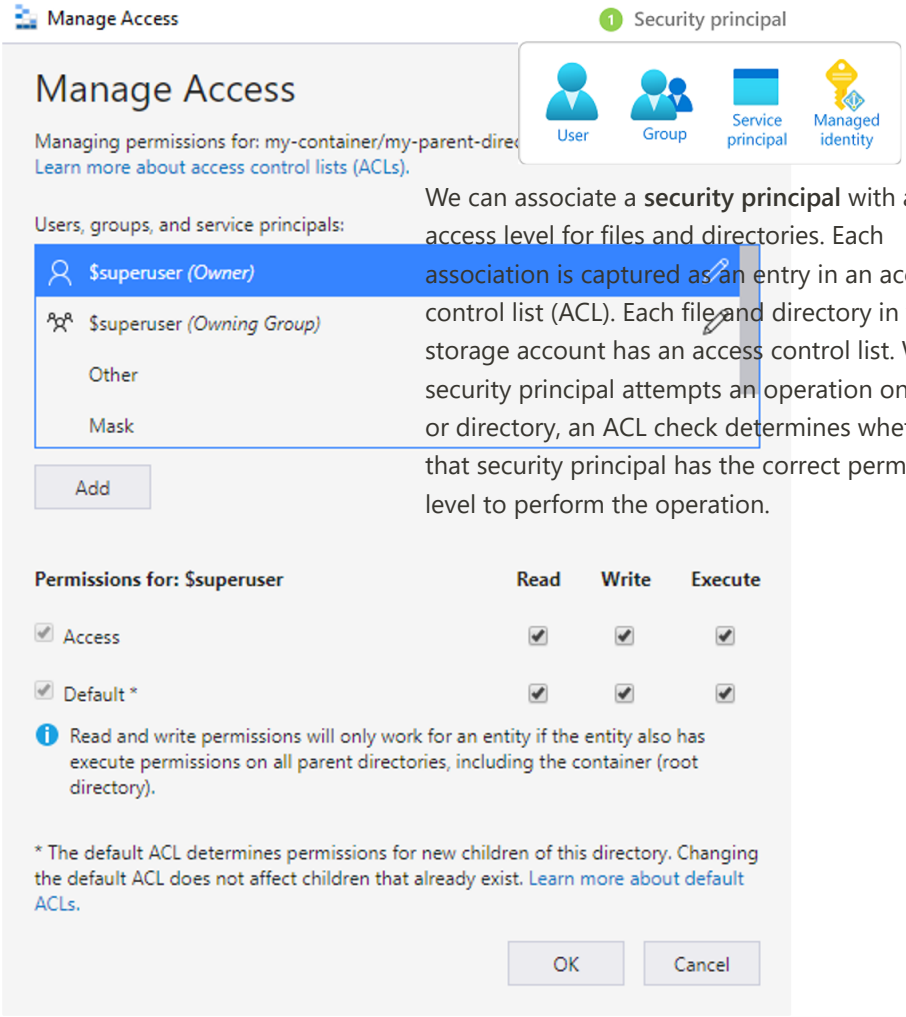
3) **Scope:** Scope is the set of resources that the access applies to.





4) Access Control Lists (ACL)

In the **POSIX-style model**, permissions for an item are stored on the item itself. In other words, permissions for an item cannot be inherited from the parent items if the permissions are set after the child item has already been created. Permissions are only inherited if default permissions have been set on the parent items before the child items have been created.



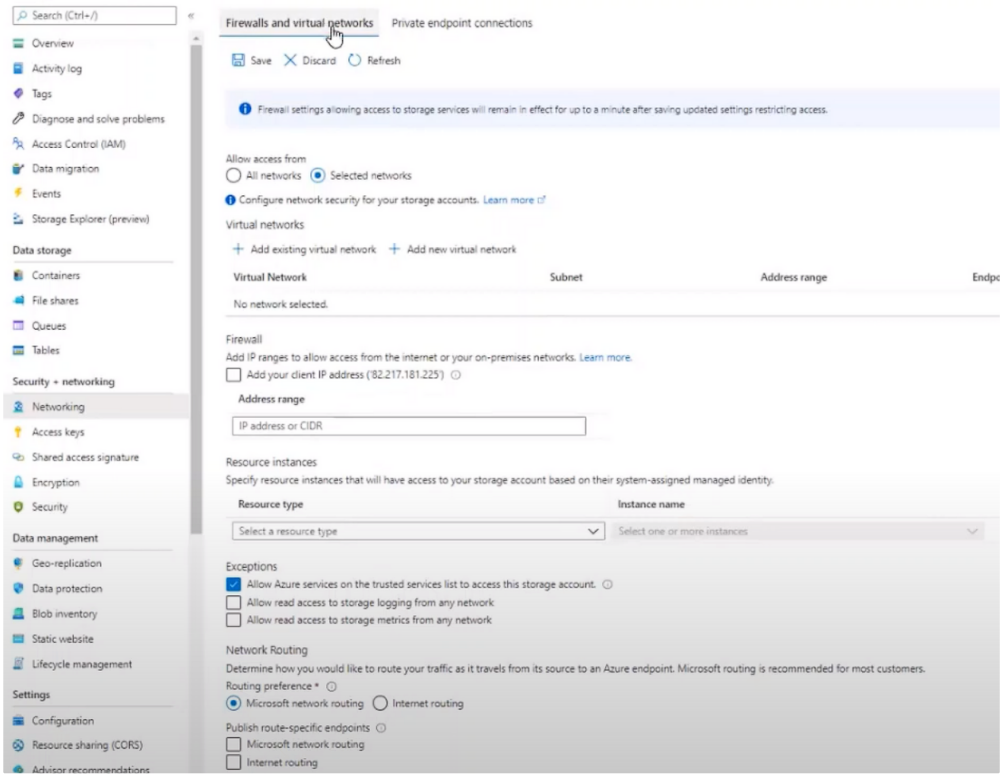
We can associate a **security principal** with an access level for files and directories. Each association is captured as an entry in an access control list (ACL). Each file and directory in your storage account has an access control list. When a security principal attempts an operation on a file or directory, an ACL check determines whether that security principal has the correct permission level to perform the operation.

📌 **ACL can never supersede an RBAC role**. It can only augment the role with additional permissions. For ex: A user who has been provided RBAC on blob storage whereas in the ACL list, has been denied all the read, write and execute permissions will still have this permission through the RBAC role.

📌 **RBAC** provides course grain permissions to the data lake or to folders inside it. These are used to allow or deny permissions to the **folder structure** but typically **do not dictate** the ability of the **user** to perform actions against the data.
ACLs are used to define the fine grain permissions to the data, this is where the ability of the user to read, write, modify or delete the data is set.

5) Firewalls and Virtual Networks

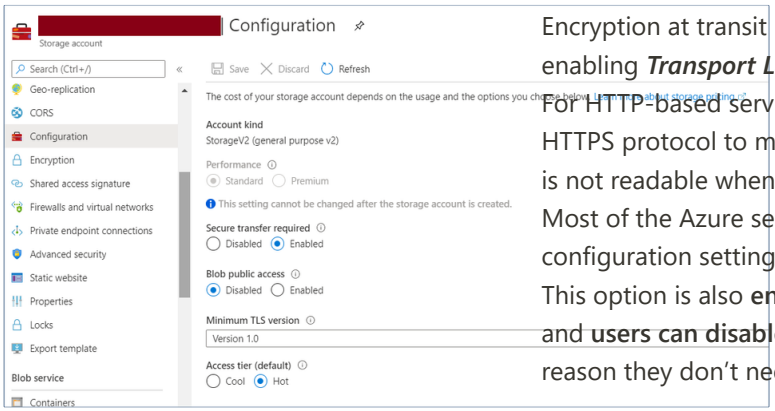
Azure Storage provides a layered security model. Storage accounts having a public endpoint is accessible through the internet. We can also create Private Endpoints for your storage account, which assigns a private IP address from our VNet to the storage account, and secures all traffic between our VNet and the storage account over a **private link**.



Authorization is supported with Azure Active Directory (Azure AD) credentials for blobs and queues, with a valid account access key, or with a SAS token. When a blob container is configured for anonymous public access, requests to read data in that container do not need to be authorized, but the firewall rules remain in effect and will block anonymous traffic.

6) Encryption at Transit

> ***Encryption at Transit refers to encrypting the data that is being moved from one place to another. Examples of data movement could be data being read by an application, data getting replicated to a different zone, or data being downloaded from the cloud.***



Secure Transfer required = Encryption at transit

Encryption at transit is achieved by enabling ***Transport Layer Security (TLS)***. For HTTP-based services, it means using HTTPS protocol to make sure that data is not readable when it is on the move. Most of the Azure services provide configuration settings to enable TLS. This option is also **enabled by default** and **users can disable** it if for any reason they don't need it.

7) Encryption at Rest

> ***Encryption at rest is the process of encrypting data before writing it to disks and decrypting the data when requested by applications.***

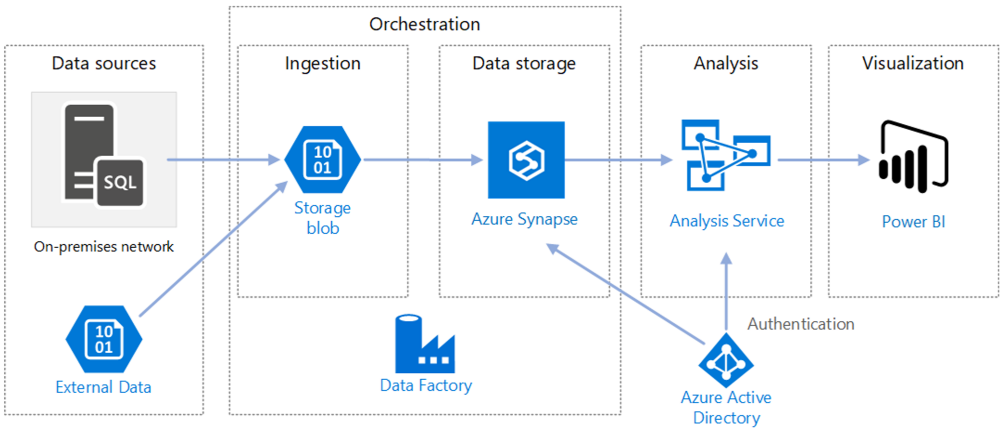
Encryption at rest is **enabled by default** and **can't be disabled**. All data written to Azure Storage is automatically encrypted by **Storage Service Encryption** (SSE) with a **256-bit Advanced Encryption Standard** (AES) cipher.

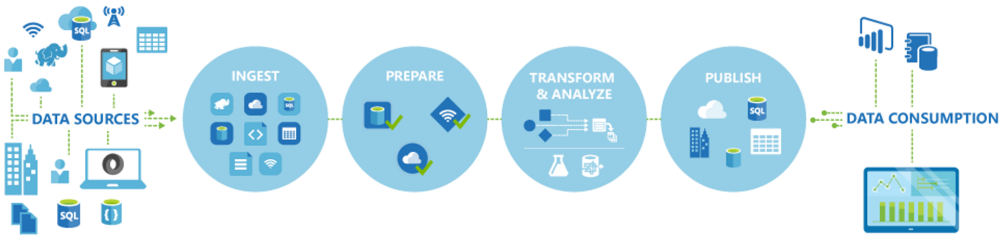
SSE automatically encrypts data when writing it to Azure Storage. When you read data from Azure Storage, Azure Storage decrypts the data before returning it. This process incurs no additional charges and doesn't degrade performance. It can't be disabled.

✅ Azure Data Factory

A good youtube video for an introduction to ADF

> ***ADF provides a cloud-based ETL solution that orchestrates data movement by scheduling data pipelines and transforming data at scale between various data stores and compute resources.***





Collect Phase

Define and connect all the required sources of data together, such as databases, file shares, and FTP web services

Transform & Analyse

Compute services such as Databricks can be used to prepare transformed data to feed prod environments with cleansed and transformed data

Publish Phase

Finally, load the data onto a destination- Azure Data Warehouse, Azure SQL Database, Azure Cosmos DB, or any other service for consumption

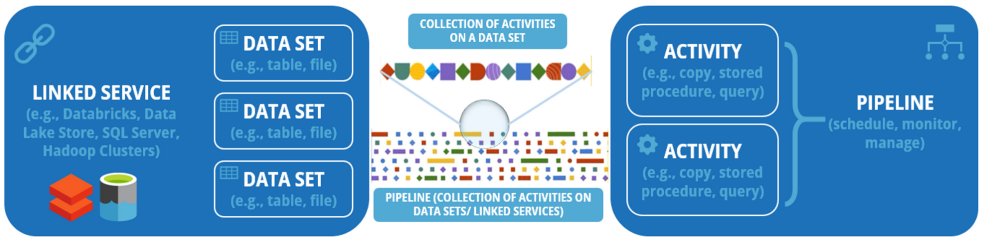
Monitor Phase

built-in support for pipeline monitoring via Azure Monitor, API, PowerShell, Azure Monitor logs, and health panels on the Azure portal

✅ ADF Top Level Concepts

Azure Data Factory is composed of key components.

→ Pipeline → Activities → Datasets → Linked Service → Data Flows
→ Integration Runtimes → Triggers → Parameters



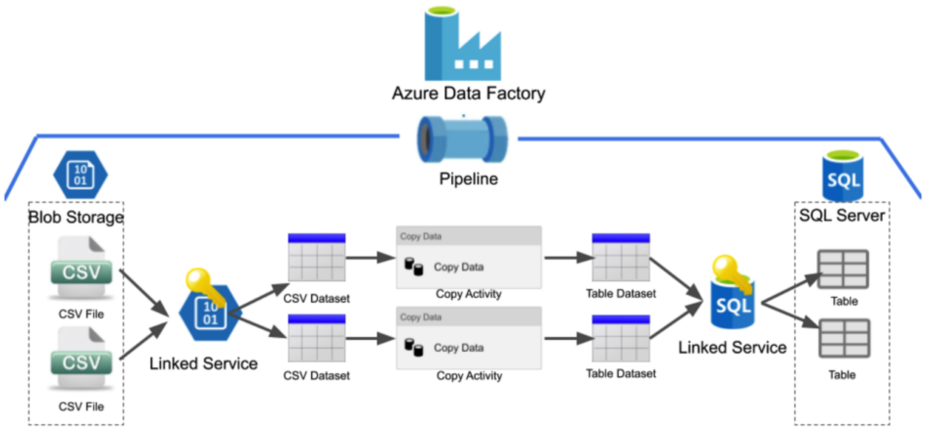
Azure Data Factory can have one or more pipelines. A **pipeline** is a logical grouping of activities that together perform a task. For example, a pipeline could contain a set of activities that ingest and clean log data, and then kick off a mapping data flow to analyze the log data. The pipeline allows you to manage the activities as a set instead of each one individually. You deploy and schedule the pipeline instead of the activities independently.

Now, a **dataset** is a named view of data that simply points or references the data you want to use in your **activities** as inputs and outputs. Before you create a dataset, you must create a **linked service** to link your data store to the Data Factory. **Linked services** are like connection strings, which define the connection information needed for the service to connect to external resources.

Think of it this way; the dataset represents the structure of the data within the linked data stores, and the linked service defines the connection to the data source. For example, to copy data from Blob storage to a SQL Database, you create two linked services: Azure Storage and Azure SQL Database. Then, create two datasets: an Azure Blob dataset (which refers to the Azure Storage linked service) and an Azure SQL Table dataset (which refers to the Azure SQL Database linked service).

In **Data Flow**, datasets are used in source and sink transformations. The datasets define the basic data schemas. If your data has no schema, you can use schema drift for your source and sink (more on schema drift later). Pipeline runs are typically instantiated by passing arguments to parameters that you define in the pipeline. You can execute a pipeline either manually or by using a ***trigger***. We have the following triggers in ADF: scheduled, tumbling window, and event-based (more on this later).

Finally, The ***Integration Runtime*** (IR) provides the compute infrastructure for completing a pipeline. We have the same three types of IR: Azure, Self-hosted, and Azure-SSIS (more on this later).

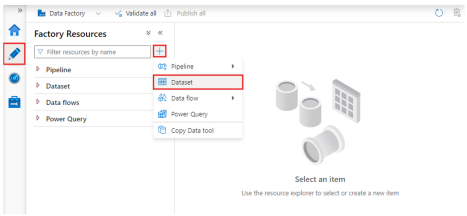


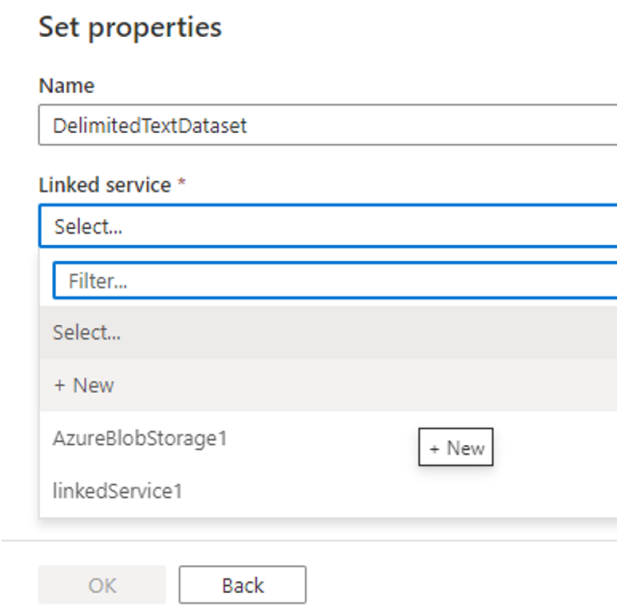
✅ Linked Service

The **Linked Service** represents the connection information that enables the ingestion of data from external resources such as a data store (Azure SQL Server) or compute service (Spark Cluster).

✅ Dataset

Datasets represent data structures within your data stores. These point to (or reference) the data that we want to use in our activities and are referenced by the Linked service.





✅ Pipeline

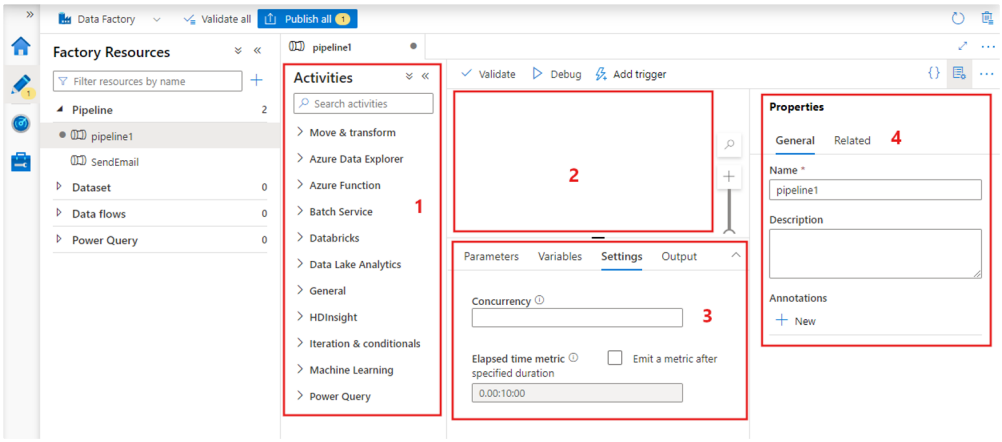
A **pipeline** represents a logical grouping of activities where the activities together perform a certain task. The advantage of using a pipeline is that you can more easily manage the activities as a set.

> 📌 **Annotations:**
> When monitoring data pipelines, you may want to be able to filter and monitor a certain group of activities, such as those of a project or specific department's pipelines. You can achieve these using annotations.
>
> Annotations are tags that you can add (only static values) to pipelines, datasets, linked services, and triggers to easily identify them. For more, click here.

✅ Activity (Inside a pipeline)

Activities are **actions** that are performed on the data. An activity can take zero or more input datasets and produce one or more output datasets. Activities contain the actual transformation logic.

An activity that depends on one or more previous activities, can have different dependency conditions. The four dependency conditions are: **Succeeded, Failed, Skipped, and Completed**.

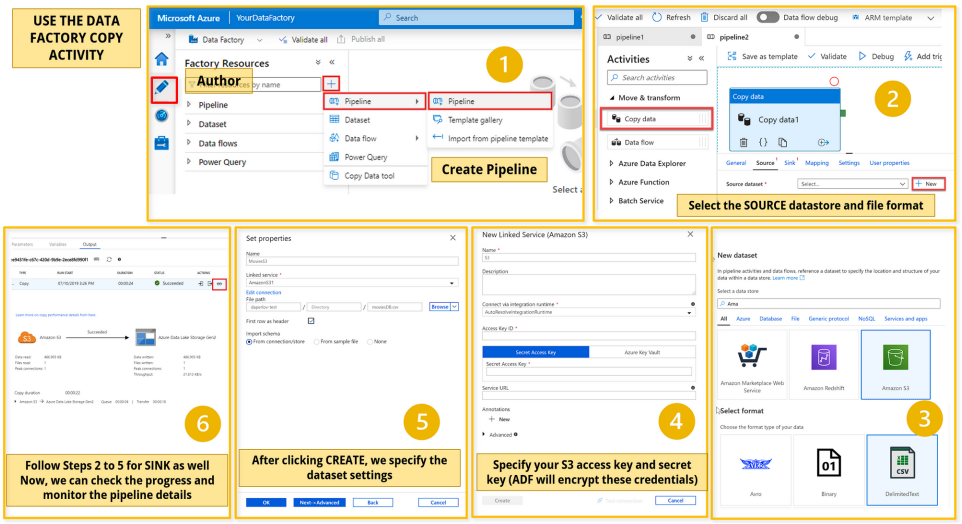


1. All activities that can be used within the pipeline.
2. The pipeline editor canvas, where activities will appear when added to the pipeline.
3. The pipeline configurations pane, including parameters, variables, general settings, and output.
4. The pipeline properties pane, where the pipeline name, optional description, and annotations can be configured. This pane will also show any related items to the pipeline within the data factory.

Because there are many activities that are possible in a pipeline in Azure Data Factory, activities can be grouped into three categories:

1) Data movement activities:

The Copy Activity in the Data Factory copies data from a source data store to a sink data store. Supported stores include all Azure offerings, selected SAP offerings, and much more. For a detailed list, click here.



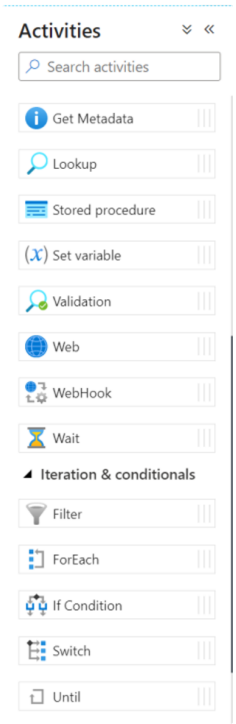
2) Data transformation activities:

Data transformation activities can be performed natively within the authoring tool of Azure Data Factory using the **Mapping Data Flow**. Alternatively, you can call a compute resource to change or enhance data through transformation, or perform analysis of the data. These include compute technologies such as Azure Databricks, Azure Batch, SQL Database and Azure Synapse Analytics. For details, click.

3) Control flow activities:

> ***Control flow*** *is a group of pipeline activities that includes chaining activities in a sequence, branching, defining parameters at the pipeline level, and passing arguments while invoking the pipeline on demand or from a trigger.*

These are activities that can affect the path of execution.



- **Append Variable**: Used to add a value to an existing array variable.
- **Set Variable:** Used to set the value of an existing variable of type String, Bool, or Array.
- **Execute Pipeline**: Allows a pipeline to invoke another pipeline.
- **If Condition:** Allows directing pipeline execution, based on evaluation of certain expressions.
- **Get Metadata**: Used to retrieve metadata of any data in ADF.
- **ForEach**: Defines a repeating control flow in your pipeline. ***ADF can start multiple activities in parallel using this approach.***
- **Lookup:** Retrieve a dataset from any of the ADF-supported data sources. **Can be used for delta loads**.
- **Filter**: Used to apply a filter expression to an input array.
- **Until:** Executes a set of activities in a loop until the condition associated with the activity evaluates to true.
- **Wait:** Wait activity allows pausing pipeline execution for the specified time period.

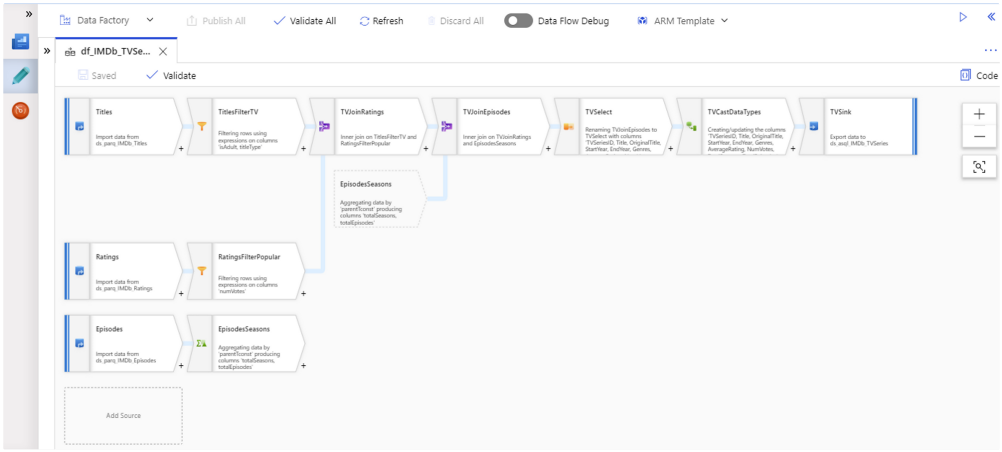
For reading more about these individual activities, click here.

✅ Data Flows

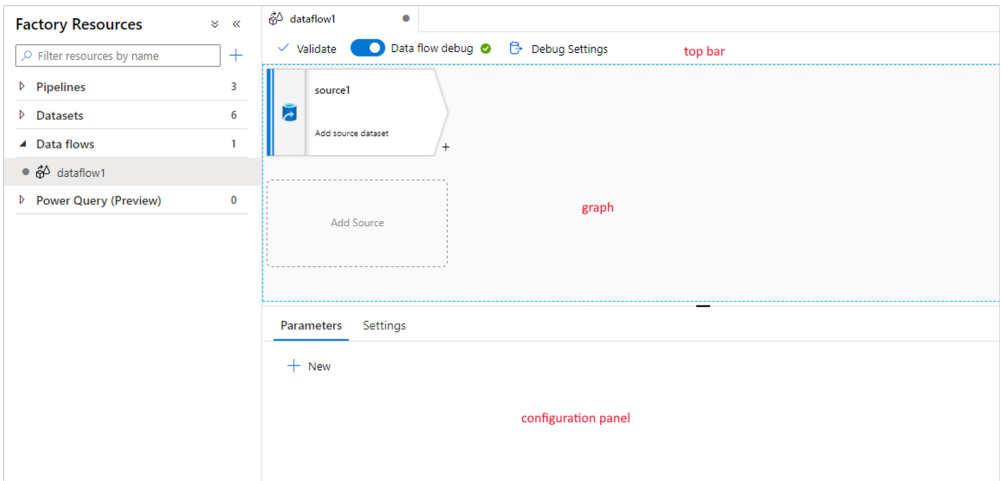
> ***Data Flows*** *are used to build code-free transformation data flows/ transformation logic that is executed on* ***automatically provisioned Apache Spark clusters****. ADF internally handles all the code translation, spark optimization, and execution of the transformation.*

Control Flow Activity	Data Flow Transformation
Affects the execution sequence or path of the pipeline	Transforms the ingested data
Can be recursive	Non-recursive
No source/sink	Source and sink are required
Implemented at the pipeline level	Implemented at the activity level

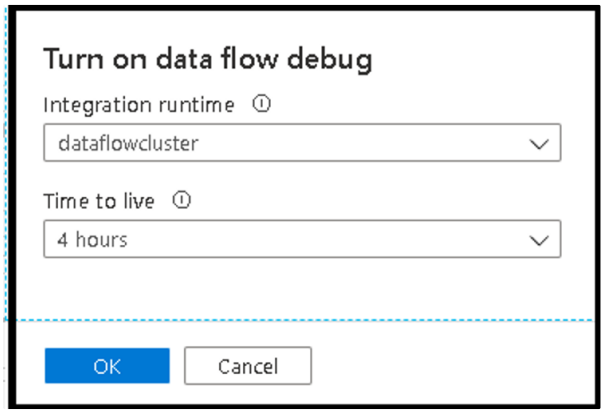
1) Transforming data using the Mapping Data Flow *(present in both ADF and Synapse Analytics)*



Data flow has a unique authoring canvas designed to make building transformation logic easy. The data flow canvas is separated into three parts: the top bar, the graph, and the configuration panel.



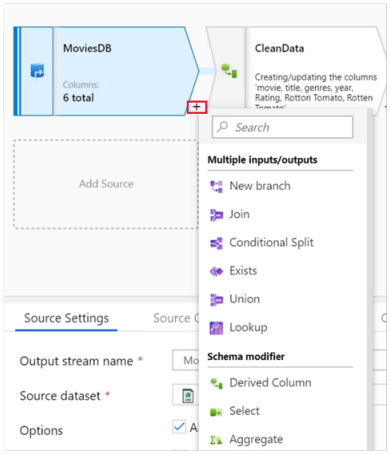
— ***Debug mode***: *Here you can actually see the results of each transformation. In the debug mode session, the data flow runs interactively on a* ***Spark cluster****. In the debug mode you will be charged on an hourly basis when the cluster is active. It typically takes 5-7 minutes for the cluster to spin up. With this mode, you are able to build your data flow step by step and view the data as it runs through each transformation phase.*



If AutoResolveIntegrationRuntime is chosen, a cluster with eight cores of general compute with a default 60-minute time to live will be spun up.

→ Graph

The graph displays the transformation stream. It shows the lineage of source data as it flows into one or more sinks. To add a new source, select **Add source**. To add a new transformation, select the plus sign on the lower right of an existing transformation.

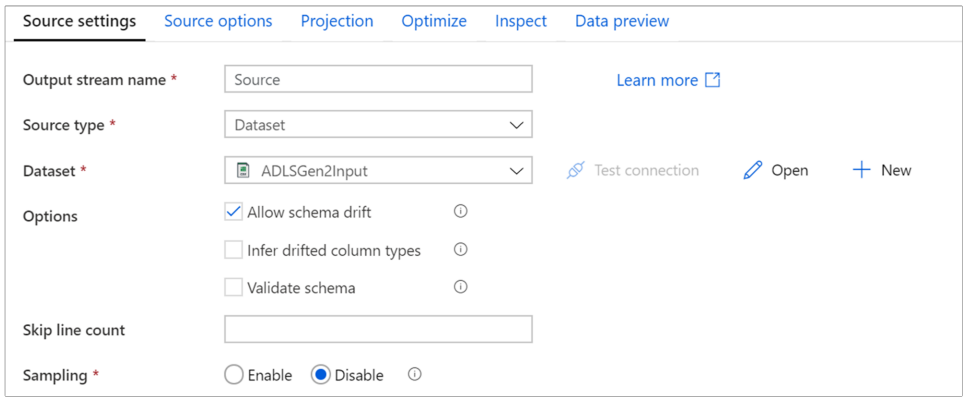


→ Configuration panel

The configuration panel shows the settings specific to the currently selected transformation. If no transformation is selected, it shows the data flow. In the overall data flow configuration, you can add parameters via the **Parameters** tab. Each transformation contains at least four configuration tabs. Read more here

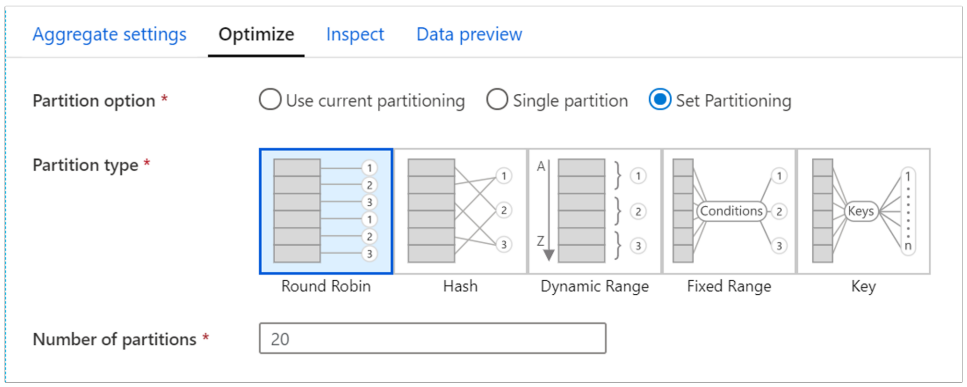
1) Transformation settings

The first tab in each transformation's configuration pane contains the settings specific to that transformation.



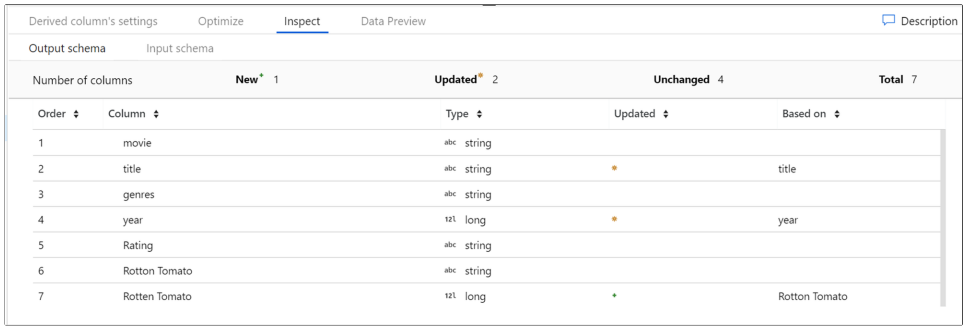
2) Optimize

The **Optimize** tab contains settings to configure partitioning schemes.



3) Inspect

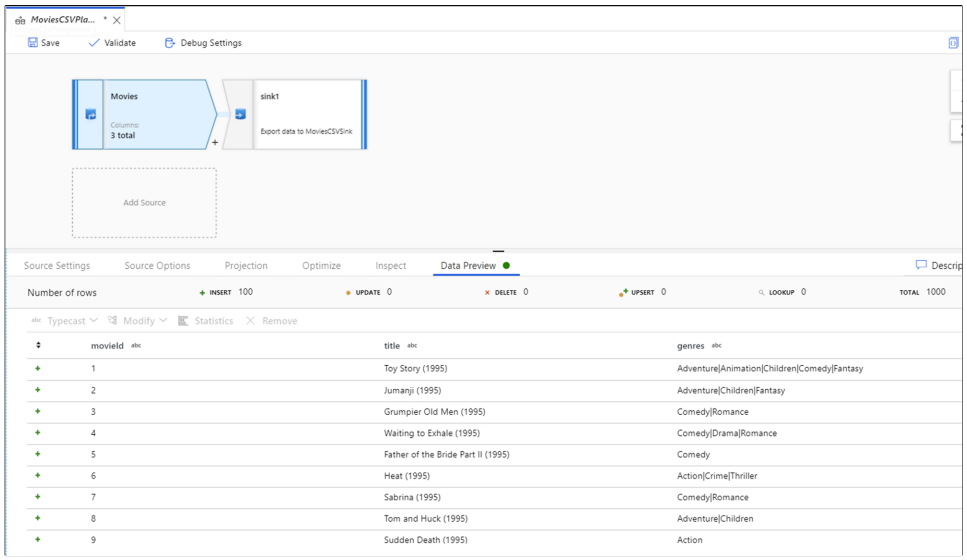
The **Inspect** tab provides a view into the metadata of the data stream that you're transforming. You can see column counts, the columns changed, the columns added, data types, the column order, and column references. **Inspect is a read-only view of your metadata**. You don't need to have debug mode enabled to see metadata in the Inspect pane.



As you change the shape of your data through transformations, you'll see the metadata changes flow in the **Inspect** pane. If there isn't a defined schema in your source transformation, then metadata won't be visible in the **Inspect** pane. Lack of metadata is common in schema drift scenarios.

4) Data preview

If debug mode is on, the **Data Preview** tab gives you an interactive snapshot of the data at each transform.



Mapping Data Flows provides a number of different transformations types that are broken down into the following categories:

Multiple input/output transformations

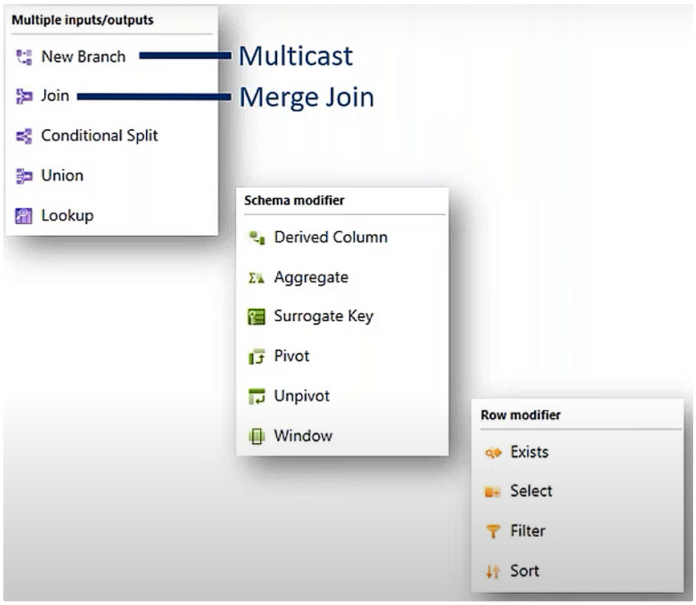
These transformations will **generate new data pipelines or merge into one** e.g. union of multiple data streams

Schema modifier transformations

Make a modification to a **sink** destination by creating **new columns** based on the action of the transformation e.g. derived column after performing some operation on the existing column

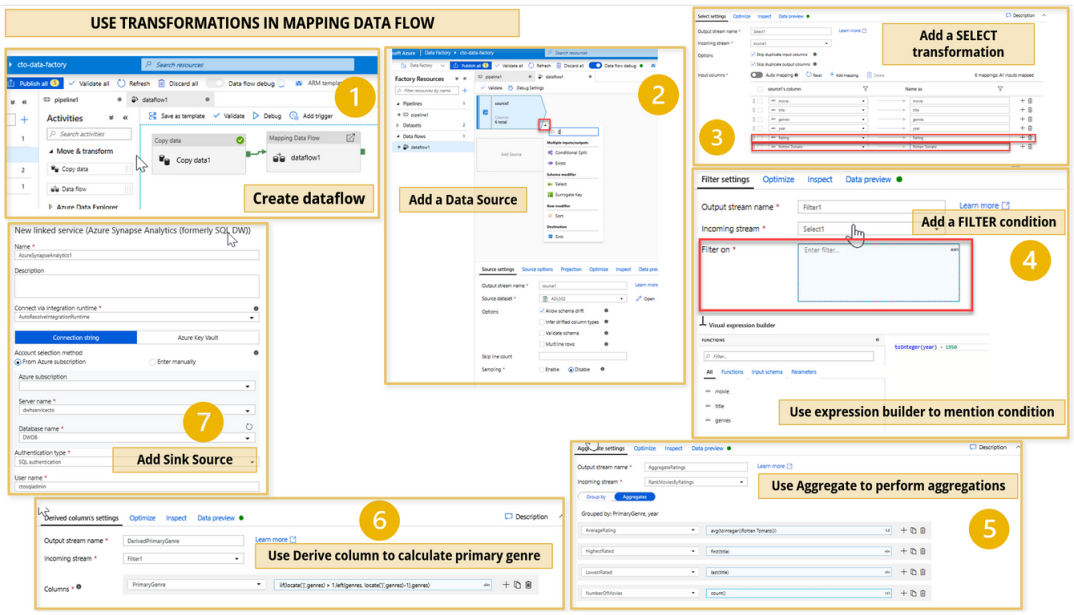
Row modifier transformations

These impact how the **rows** are presented in the **sink** e.g. sorting of a particular column



Note: Filter transformation in data flow is different from Filter activity in control flow

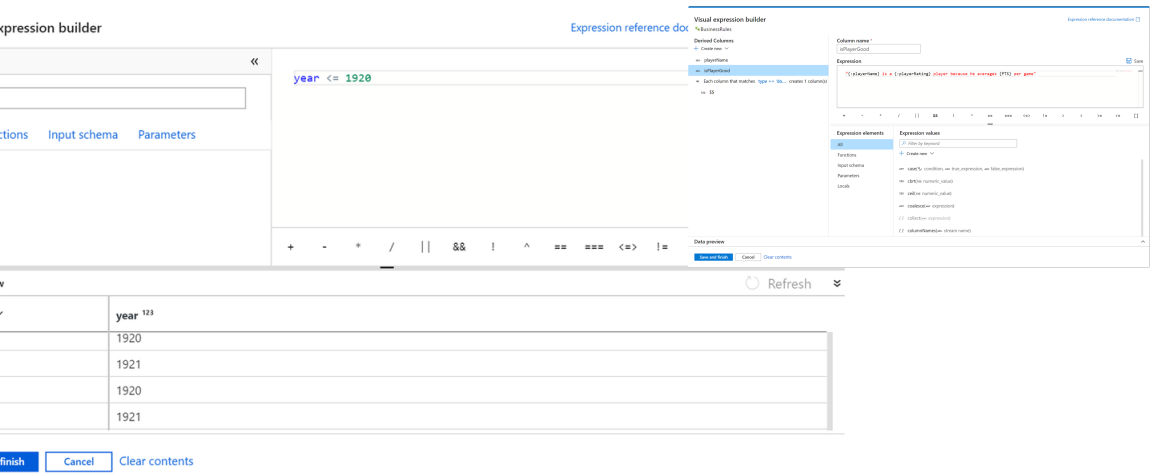
Example of Mapping Data Flow



Link showing the detailed implementation steps or here. To learn about optimizing data flow, check this link.

1.a) Data Flow Expression Builder

Some of the transformations can be defined using a **Data Flow Expression Builder** that will enable you to customize the functionality of a transformation using columns, fields, variables, parameters, and functions from your data flow in these boxes.

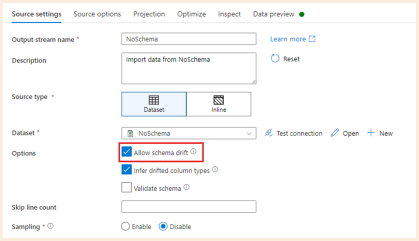


Here is a sample expression that can be used to create date directories and automatic partitioning:

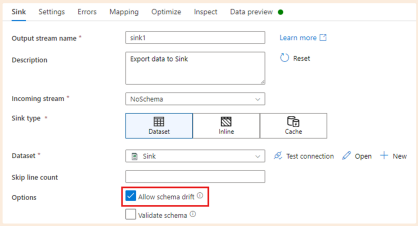
```
"staging/driver/out/" + toString(year(currentDate()))  + "/" + toString(m
onth(currentDate())) + "/" + toString(dayOfMonth(currentDate()))
```

Schema Drift

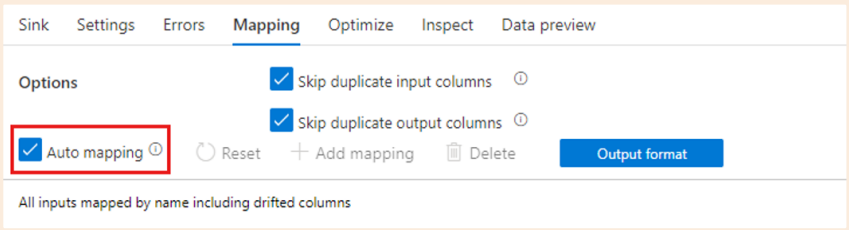
Schema drift is the case where your sources often change metadata. Fields, columns, and, types can be added, removed, or changed on the fly. Without handling schema drift, your data flow becomes vulnerable to upstream data source changes.



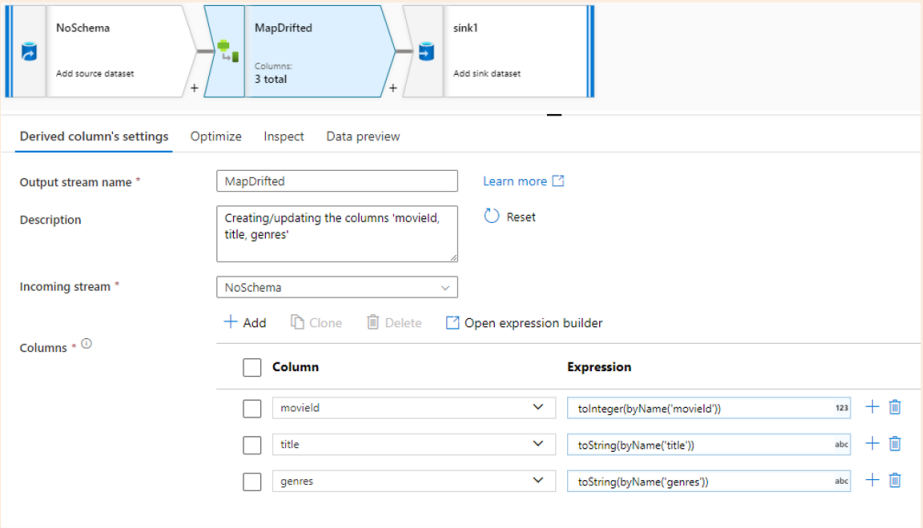
Schema drift in Source



Schema drift in Sink



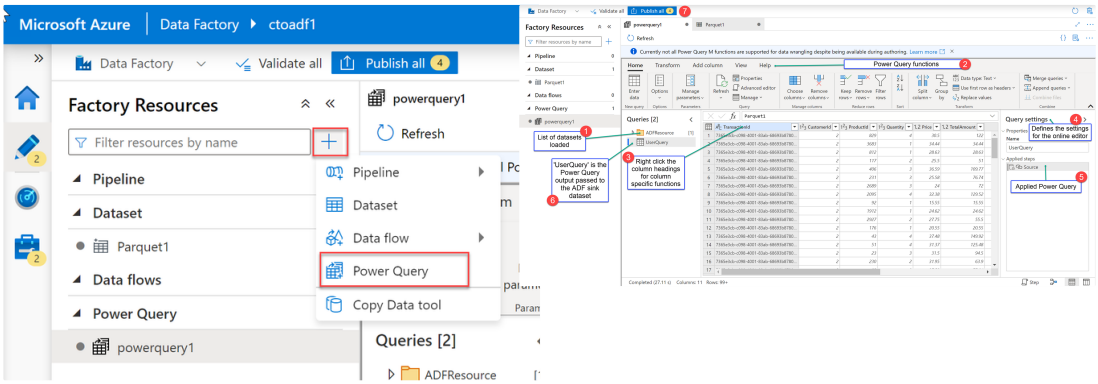
If schema drift is enabled, make sure the Auto-mapping slider in the Mapping tab is turned on. With this slider on, all incoming columns are written to your destination. Otherwise, you must use rule-based mapping to write drifted columns.



In the Derived Column transformation, each drifted column is mapped to its detected name and data type

2) Transforming data using Wrangling Data Flows *(ADF only, not present in Synapse Analytics)*

Wrangling data flow is used for data prepping using Power Query



✅ Parameters

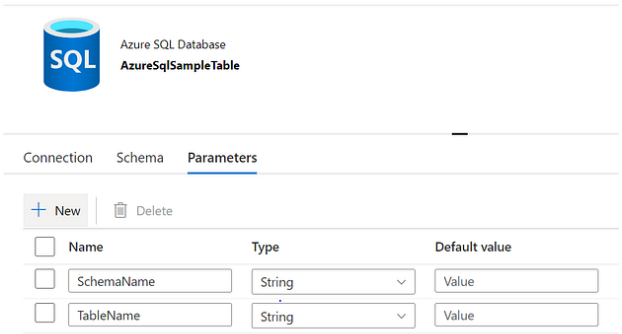
> ***Parameters are used to pass external values into pipelines, datasets, linked services, and data flows. These are key-value pairs of read-only configuration.*** *Once the parameter has been passed into the resource, it cannot be changed. By parameterizing resources, you can reuse them with different values each time. This reduces redundancy in your ETL pipelines and improves flexibility.*

1) Dataset Parameters

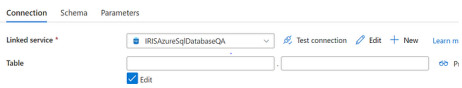
When working with a database with multiple tables in it, instead of creating a new dataset for using each of them, dataset parameters can be used to pass the table names at run time.

- You will need to create a dataset without mentioning the table name while creating it

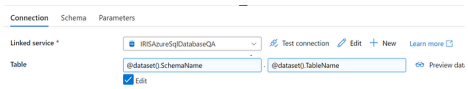
- Parameters have to be added in the parameters tab. In the example for creating the Azure SQL dataset, we have added two parameters, one for the schema name and the other for the name of the table.



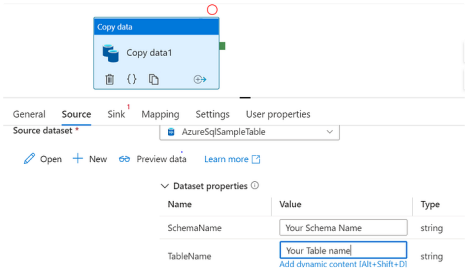
- Click on edit below the table. Then we can click on add dynamic content which will appear below the table. After clicking on that we will be able to see the parameters we added.



- We can click on the parameters we want to add by clicking on them. It will look like this



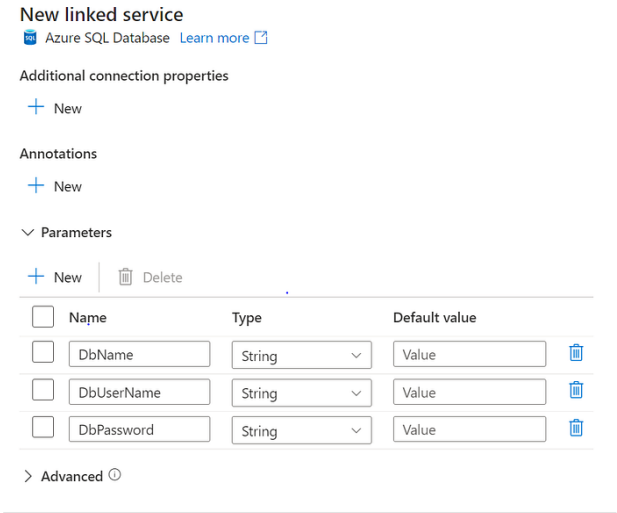
- After saving this dataset, You can pass table name and schema names parameters inside the pipeline. ADF will ask for these values when we trigger/debug the pipeline.



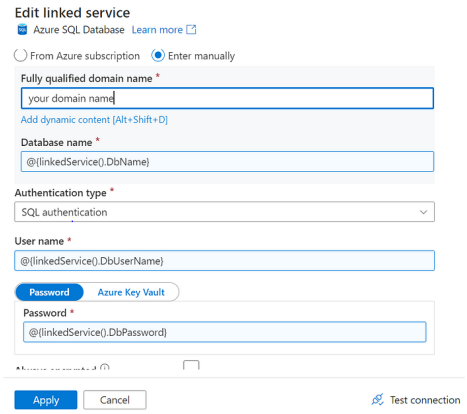
2) Linked Service Parameters

Linked service parameters can be used to parameterize the domain name, database name, username, and password for the database.

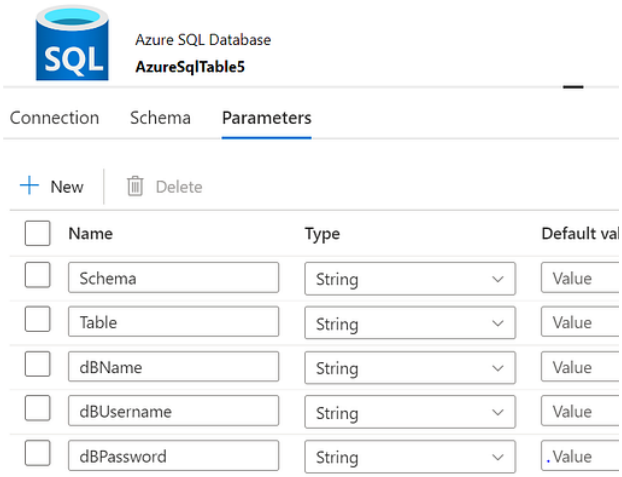
- In the following example, we are creating a new Azure SQL database. Go to the linked service in the monitor tab of ADF and create a new linked service. When we create a new linked service, we will see the option of parameters, scrolling down to the bottom.



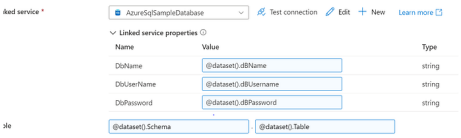
- These can be used in the linked service connection. In order to use these parameters whenever we create a new dataset, we can create dataset parameters.



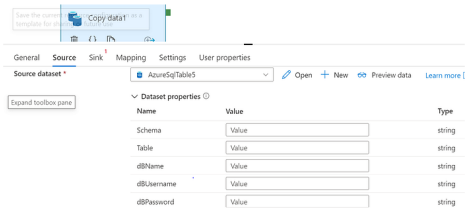
- Here parameters are added while creating the Azure SQL dataset, to pass the values to the linked service



- In the linked service itself, if we select the linked service with parameters in it, we will see Linked service properties, where we can either hardcode the values or pass dataset parameters to use them at the run time.

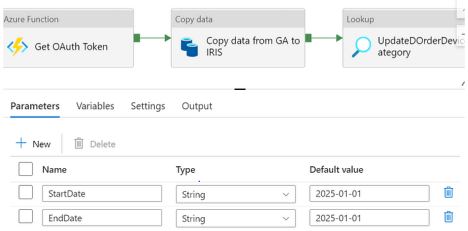


- We will see the dataset properties in the pipeline where we can pass the values.



3) Pipeline Parameters

- Pipeline parameters can be created by clicking on the blank space in the pipeline
- These can be accessed by using the syntax `@pipeline().parameters.<parameter name>`



4) Parameters in Mapping Data Flow

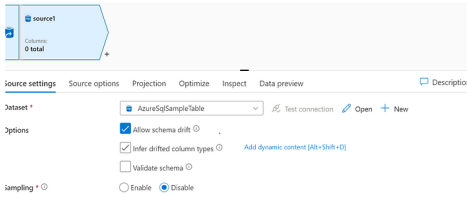
There are three options for setting the values in the data flow activity expressions:

- Use the pipeline control flow expression language to set a dynamic value.
- Use the data flow expression language to set a dynamic value.
- Use either expression language to set a static literal value.

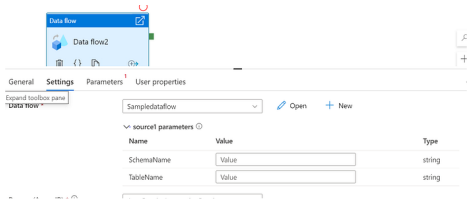
The reason for parameterizing mapping data flows is to make sure that your data flows are generalized, flexible, and reusable.

- Data flow is one of the activities in ADF pipeline, so the way to pass the parameters to it is the same as passing pipeline parameters above.

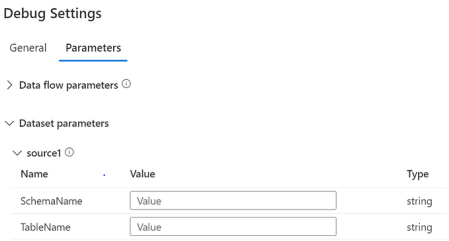
- When we create a dataflow we can select any parameterized dataset, for example, we have selected the dataset from the DATASET PARAMETERS section below.



- Now when we add dataflow activity in the pipeline and select the above dataflow, we will see the source1 parameters option to pass the table name and schema name.



- If we want to access these during debugging the dataflow and not in the pipeline itself, we can see debug settings beside it. Inside the setting, we have the option to define the parameters.



Link to an example showing Integration of a Notebook within Azure Synapse Pipelines

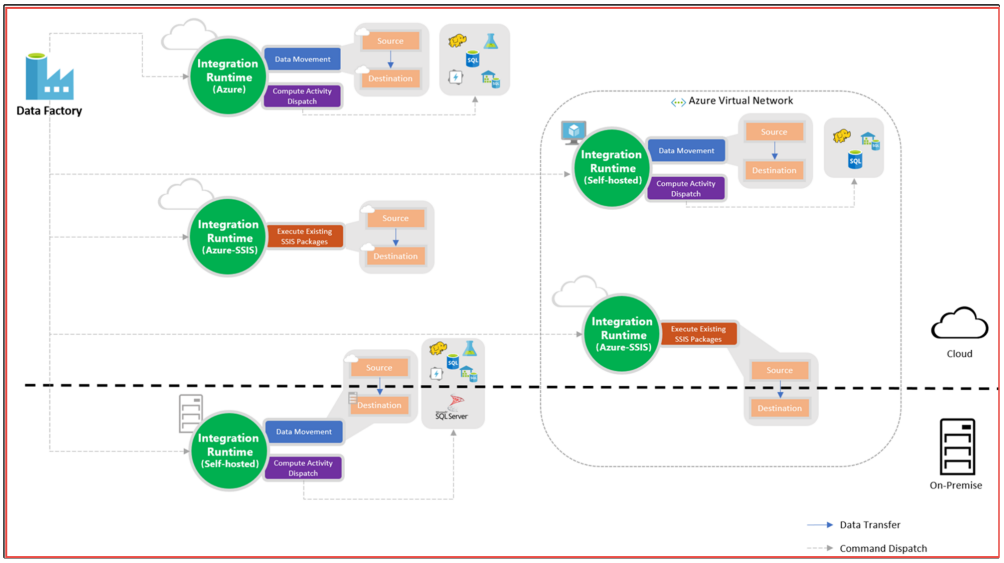
✅ Integration Runtime

ADF is a managed service (PaaS) i.e it will create the required computing infrastructure to complete the activity. This is known as ***integration runtime***. Thus IR provides a fully managed, serverless computing infrastructure. There are three types of Integration Runtime which are discussed later.

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

- **Data Flow**: Execute a Data Flow in a managed Azure compute environment.
- **Data movement**: Copy data across data stores in a public or private network (for both on-premises or virtual private networks).

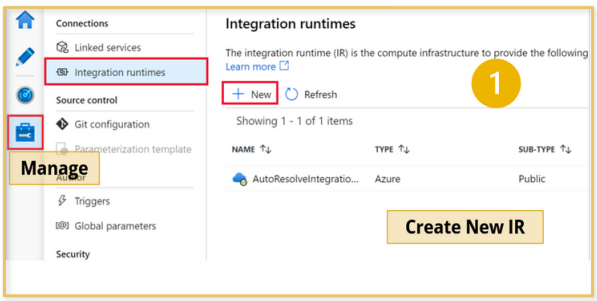
- **Activity dispatch**: Dispatch and monitor transformation activities running on a variety of compute services such as Azure Databricks, Azure HDInsight, ML Studio (classic), Azure SQL Database, SQL Server, and more.
- **SSIS package execution**: Natively execute SQL Server Integration Services (SSIS) packages in a managed Azure compute environment.

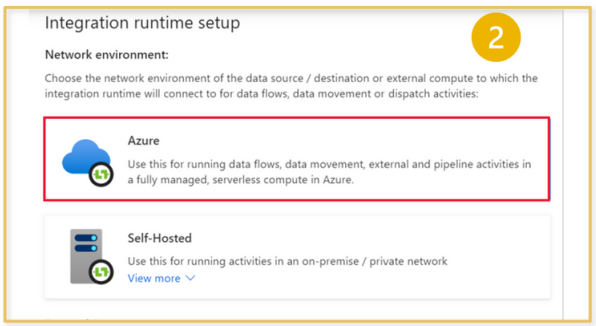


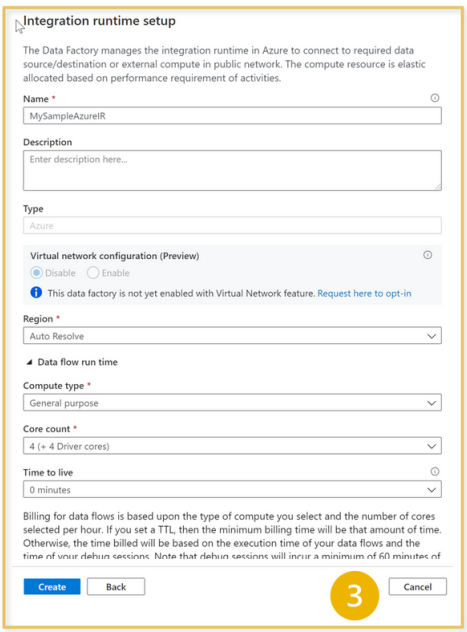
1) Azure Integration Runtime

> *Works on public networks.*
> *Provides Data Flow, Data movement and Activity dispatch*

AZURE INTEGRATION RUNTIME



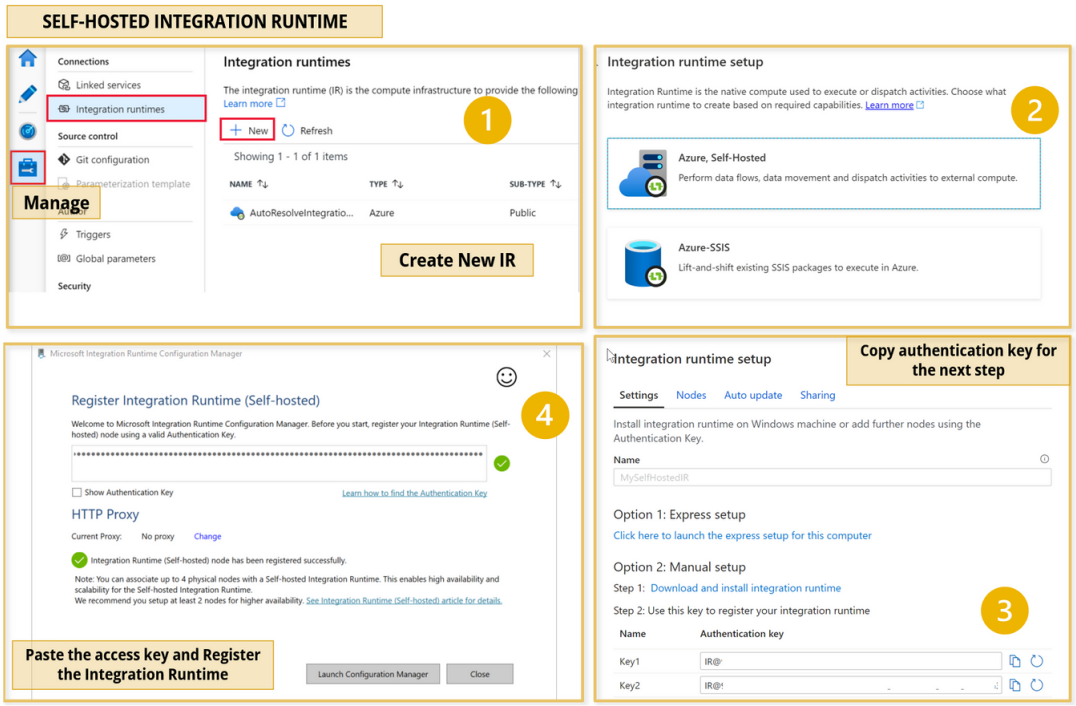




There is **auto-resolve Azure IR** option that automatically detects the sink and source data store to choose the best location either in the same region if available or the closest one in the same geography. Its best to avoid this feature and manually enter the locations.

2) Self-hosted Integration Runtime

> *Works on public and private networks*
> *Provides Data movement and Activity dispatch*



The **self-hosted integration runtime** is logically registered to the Azure Data Factory and the compute resource used to support its function is provided by you. Therefore there is no explicit location property for self-hosted IR. In order to use the on-premise infrastructure, we need to register the server and install the self-hosted IR.

3) Azure SSIS Integration Runtime

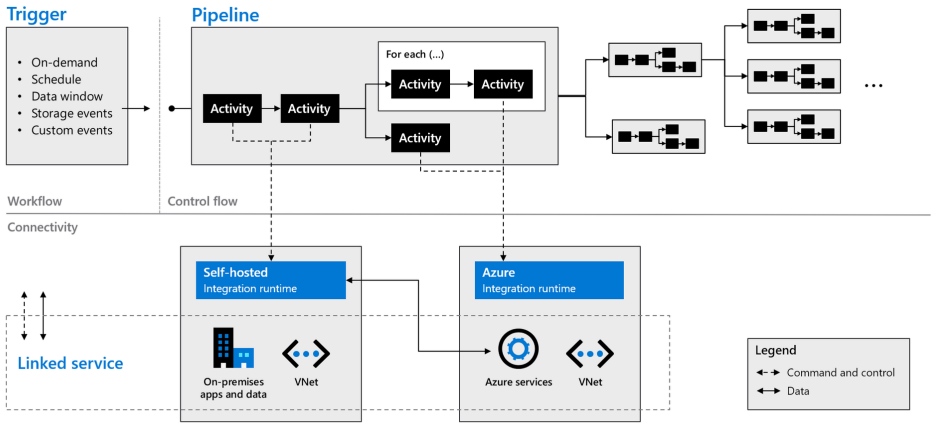
> *Works on public and private networks*
> *Supports SSIS package execution*

The Azure-SSIS IR is a fully managed cluster of Azure VMs dedicated to running your SSIS packages.

Link to an example showing the detailed implementation steps

✅ Triggers

> *Triggers are used to schedule a Data Pipeline runs without any interventions. In other words, it's a processing unit that determines when to begin or invoke an end-to-end pipeline execution*

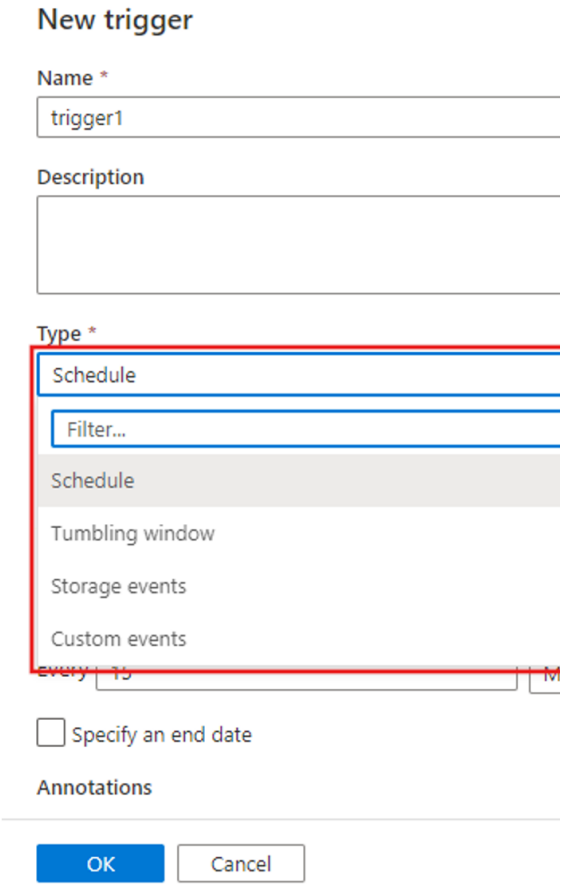


Azure Data Factory Triggers come in **three** different types: Schedule Trigger, Tumbling Window Trigger, and Event-based Trigger.

1) Schedule Trigger

This Azure Data Factory Trigger is a popular trigger that can run a Data Pipeline according to a **predetermined schedule**. It provides extra flexibility by allowing for different scheduling intervals like a minute(s), hour(s), day(s), week(s), or month(s).

The Schedule Azure Data Factory Triggers are built with a "**many to many**" **relationship** in mind, which implies that one Schedule Trigger can run several Data Pipelines, and a single Data Pipeline can be run by multiple Schedule Triggers.

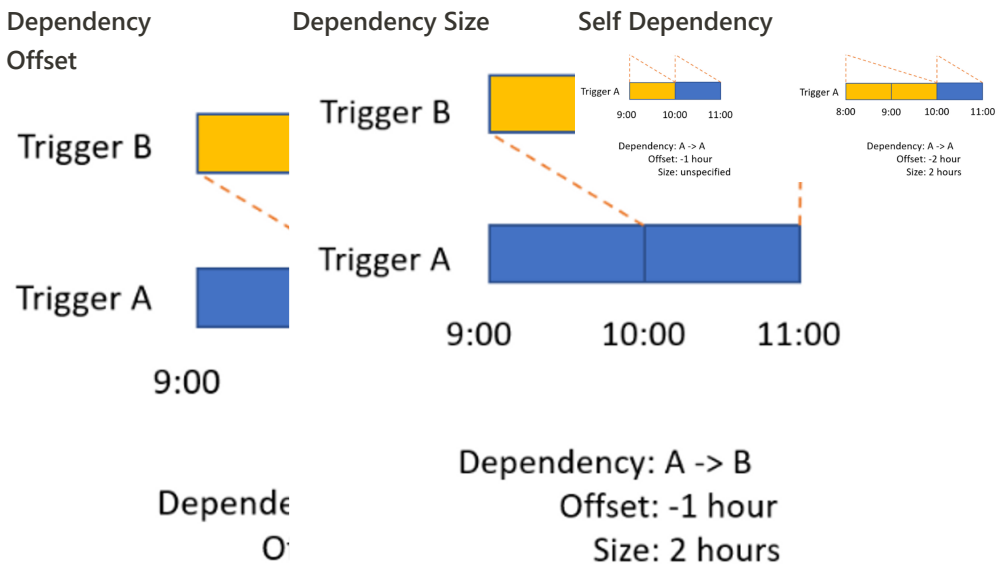


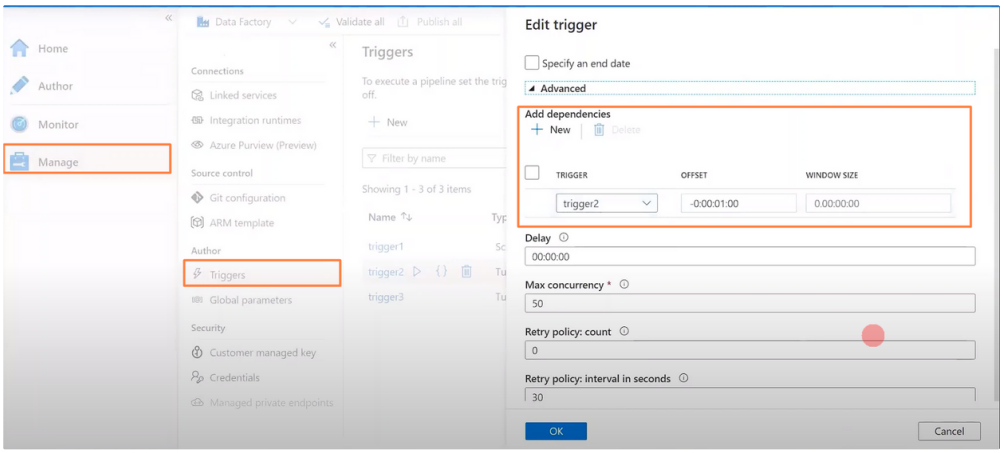
2) Tumbling Window Trigger

The Tumbling Window Azure Data Factory Trigger executes Data Pipelines at a **specified time slice** or **pre-determined periodic time interval**. It is significantly more advantageous than Schedule Triggers when working with historical data to copy or migrate data.

Consider the scenario in which you need to replicate data from a Database into a Data Lake on a regular basis, and you want to keep it in separate files or folders for every hour or day.

To implement this use case, you have to set a Tumbling Window Azure Data Factory Trigger for every 1 hour or every 24 hours. The Tumbling Window Trigger sends the start and end times for each time window to the Database, returning all data between those periods. Finally, the data for each hour or day can be saved in its own file or folder.

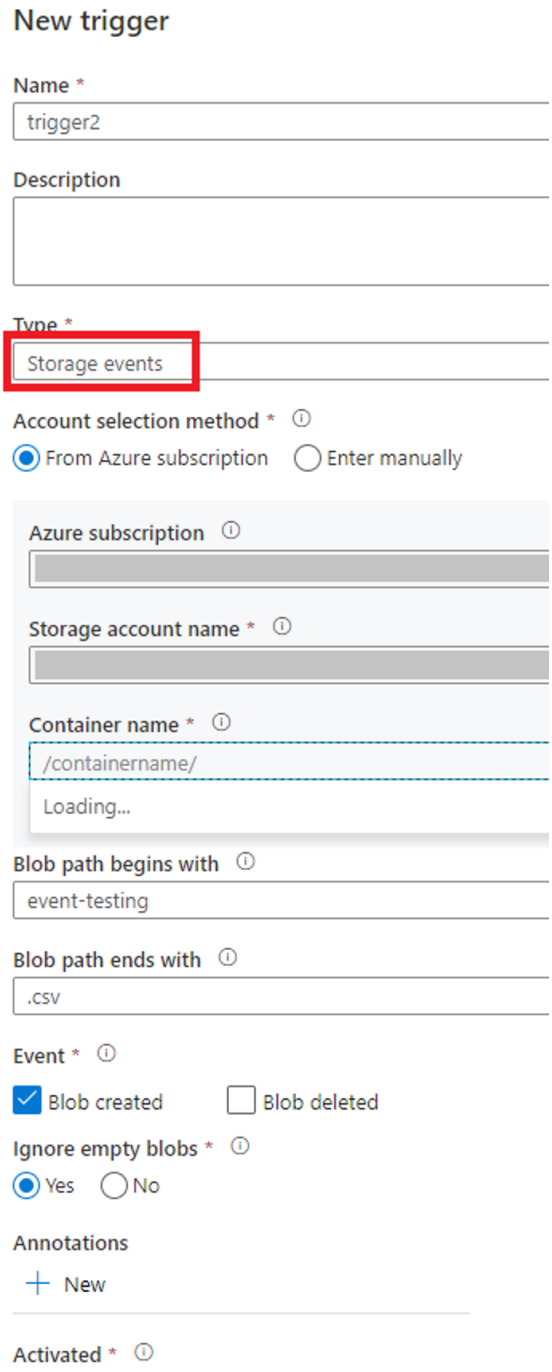




3) Event-based Trigger

The Event-based Azure Data Factory Trigger runs Data Pipelines i**n response to blob-related events**, such as generating or deleting a blob file present in Azure Blob Storage.

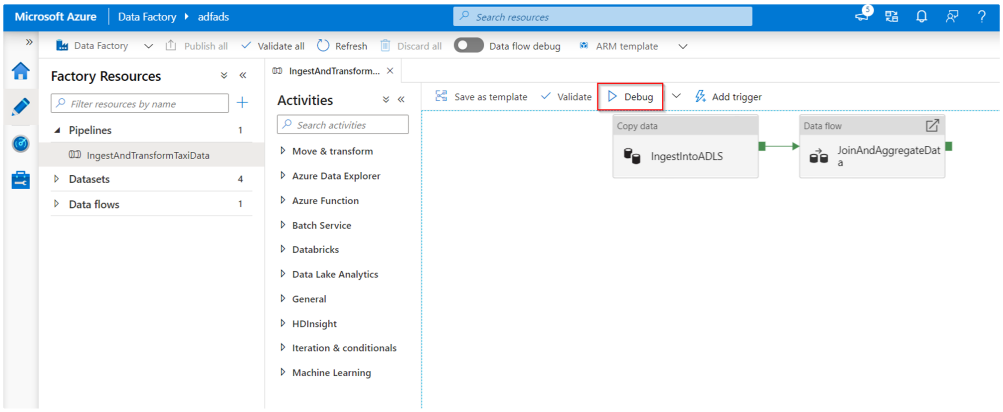
In addition, Event-based Triggers are not only compatible with blob, but also with Azure Data lake Storage. Event Triggers also work on **many-to-many relationships**, in which a single Event Trigger can run several Pipelines, and a single Pipeline can be run by multiple Event Triggers



✅ Debug and Publish a pipeline

(link - introduction to debugging provided in mapping data flow)

Azure Data Factory can help iteratively debug Data Factory pipelines when developing data integration solutions. You don't need to publish changes in the pipeline or activities before you debug. This is helpful in a scenario where you want to test the changes and see if it works as expected before you actually save and publish them.



Sometimes, you don't want to debug the whole pipeline but test a part of the pipeline. You can test the pipeline end to end or set a breakpoint. By doing so in debug mode, you can interactively see the results of each step while you build and debug your pipeline.

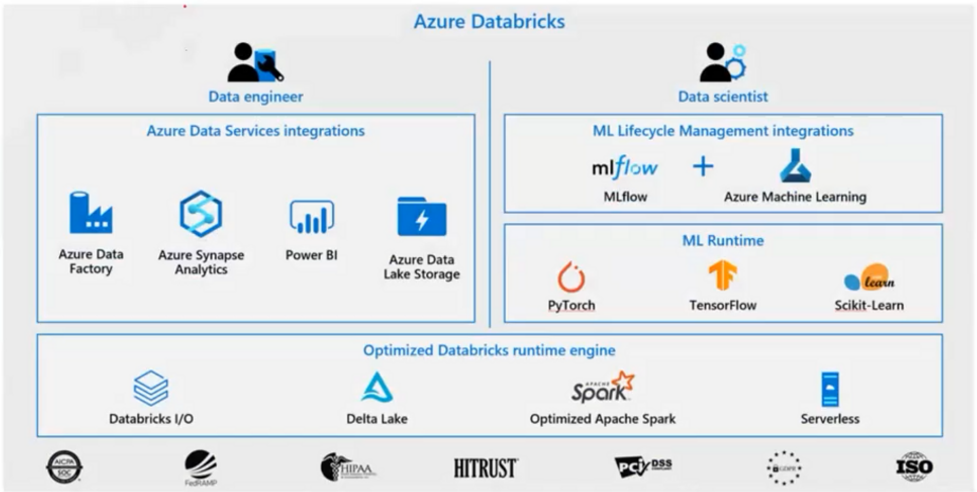
✅ Manage Source Control (CI/CD)

Azure Data Factory integrates with Azure DevOps and GitHub to allow easy source control and effective continuous integration and delivery. Azure Data Factory also offers a variety of both visual and programmatic monitoring services to also support the monitoring of your pipelines.

Link showing the detailed implementation steps

✅ Azure Databricks

> ***Databricks is a comprehensive data analytics solution built on Apache Spark and offers native SQL capabilities as well as workload-optimized Spark clusters for data analytics and data science. Databricks provides an interactive user interface through which the system can be managed and data can be explored in interactive notebooks.***



What is Apache Spark

Apache Spark emerged to provide a parallel processing framework that supports in-memory processing to boost the performance of big-data analytical applications on massive volumes of data

Interactive Data Analysis:
Used by business analysts or data engineers to analyze and prepare data

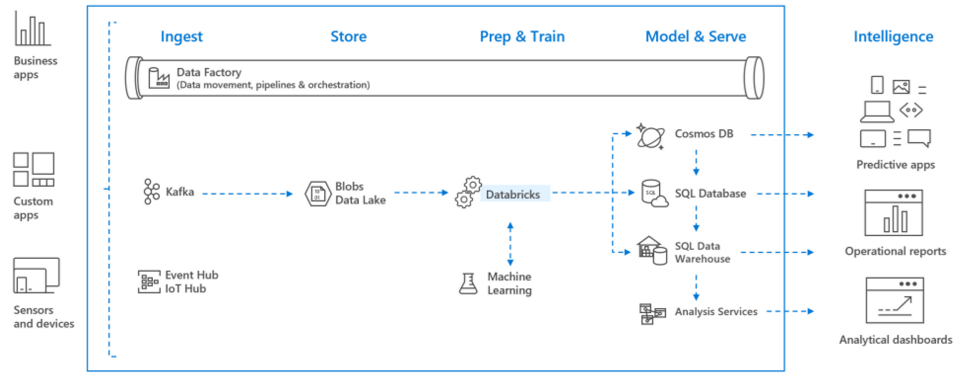
Streaming Analytics:
Ingest data from technologies such as Kafka and Flume to ingest data in real-time

Machine Learning:
Contains a number of libraries that enables a Data Scientist to perform Machine Learning

Why use Azure Databricks?
Azure Databricks is a wrapper around Apache Spark that simplifies the provisioning and configuration of a Spark cluster in a GUI interface

Azure Databricks components:
Spark SQL and DataFrames
Streaming
Mlib
GraphX
Spark Core API

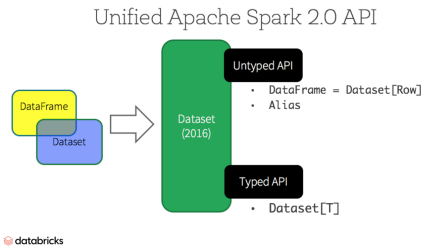
Remember that ***Spark is a replacement for MapReduce, not Hadoop***. *It's a part of the ecosystem.* Thus, Spark requires two more things to work: **Storage** (local storage/HDFS/Amazon S3) and **Resource Manager** (YARN/Mesos/Kubernetes). Spark is written in SCALA but it officially supports Java, **Scala (most used)**, **Python (PySpark),** and R.



Azure Databricks Architecture
Databricks can process the data from ADLS without importing the data into Databricks by mounting on it

Apache Spark supports data transformations with three different Application Programming Interfaces (APIs): Resilient Distributed Datasets (RDDs), DataFrames, and Datasets.

However, in the latest version, dataset and dataframe are combined to be called a dataset.



Azure Databricks is an amalgamation of multiple technologies that enable you to work with data at scale.

Workspace

It is an environment for accessing all of Azure Databricks assets. The workspace organizes objects such as notebooks, libraries, queries, and dashboards into folders, and provides access to data and computational resources such as clusters and jobs. Each workspace is isolated from others and each workspace has its own identifier.

Databricks File System (DBFS)

DBFS is a filesystem abstraction layer over a blob store. While each cluster node has its own local file system (on which the operating system and other node-specific files are stored), the cluster nodes also have access to a shared, distributed file system that they can access and operate on. The ***Databricks File System*** (DBFS) enables you to **mount cloud storage** and use it to work with **persistent file-based data.**

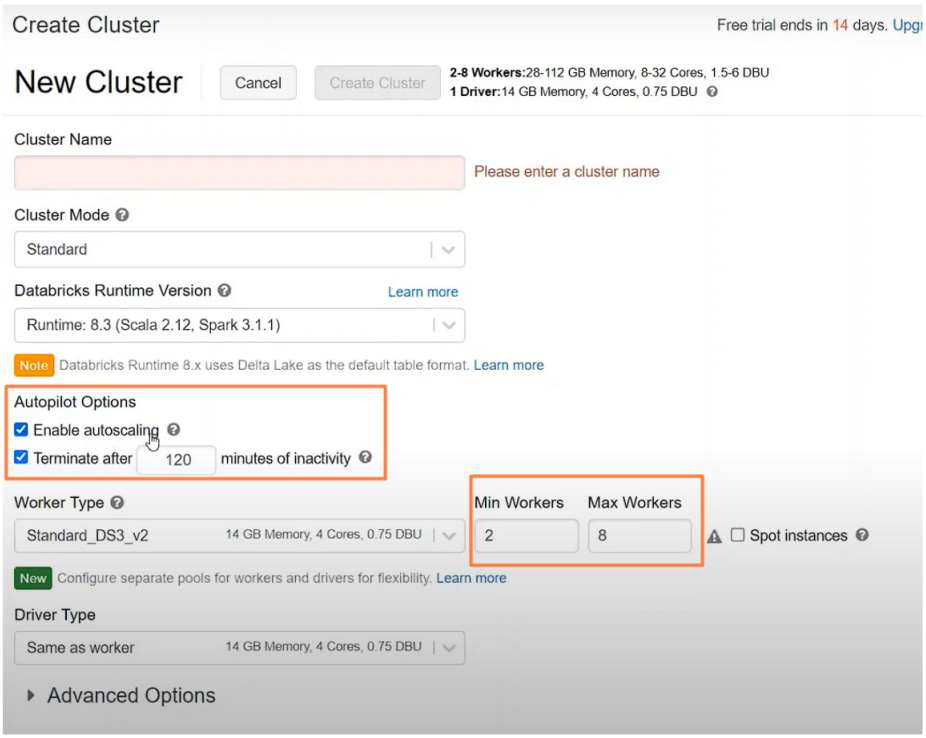
Apache Spark clusters

Spark is a distributed data processing solution that makes use of ***clusters*** to scale processing across multiple compute nodes. Each Spark cluster has a ***driver*** node to coordinate processing jobs and one or more ***worker*** nodes on which the processing occurs. This distributed model enables each node to operate on a subset of the job in parallel; reducing the overall time for the job to complete.

1) Interactive Cluster-

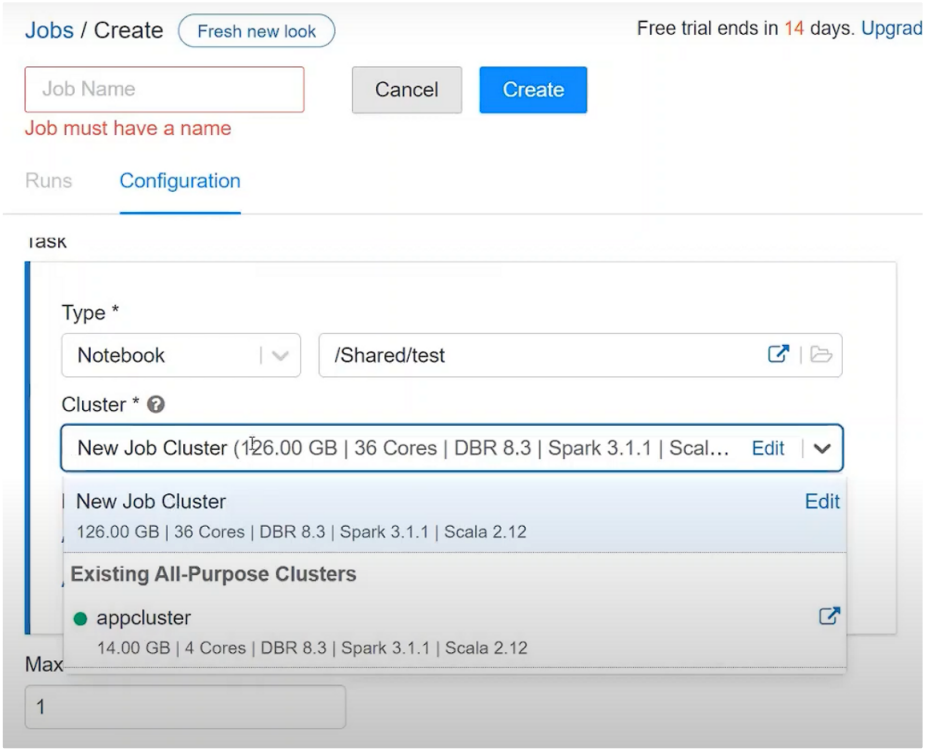
Multiple users can interactively analyze the data together. Need to terminate the cluster after job completion. These are comparatively costly and can autoscale on demand.

1. **Standard Cluster Mode**- This is used for **single-user** use, and provides no fault isolation. Supports **Scala**, Python, SQL, R, and **Java**.
2. **High Concurrency Cluster Mode**- This is used for multiple users, and provides fault isolation along with maximum cluster utilization. Supports **only Python, SQL & R.** The performance, security, and fault isolation of high concurrency clusters is provided by running user code in separate processes, which is not possible in Scala.



2) Automated/Job Cluster-

These are auto-created and auto-terminated for running automated jobs. These provide high throughput with auto-scaling capability although being comparatively cheaper.



Notebooks

One of the most common ways to work with Spark is by writing code in notebooks. Notebooks provide an interactive environment in which you can combine text and graphics in Markdown format with cells containing code that you run interactively in the notebook session.

Hive metastore

Hive is an open-source technology used to define a relational abstraction layer of tables over file-based data. The tables can then be queried using SQL syntax. **The table definitions and details of the file system locations on which they're based are stored in the metastore for a Spark cluster.** A ***Hive metastore*** is created for each cluster when it's created, but you can configure a cluster to use an existing external metastore if necessary.

Delta Lake

Delta Lake builds on the relational table schema abstraction over files in the data lake to **add support for SQL semantics** commonly found in relational database systems. Capabilities provided by Delta Lake include transaction logging, data type constraints, and the ability to incorporate streaming data into a relational table.

SQL Warehouses

SQL Warehouses are relational compute resources with endpoints that enable client applications to connect to an Azure Databricks workspace and use SQL to work with data in tables. **SQL Warehouses are only available in premium tier Azure Databricks workspaces**.

✅ Internal working

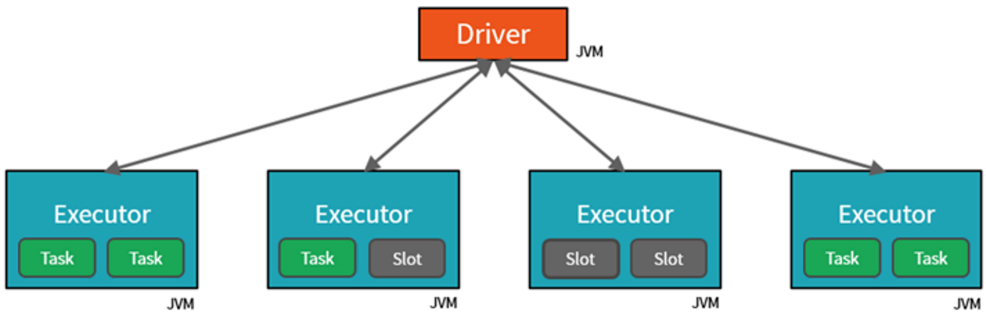
In Databricks, the notebook interface is typically the driver program. ***SparkContext***, an object of the driver program runs the main function, creates distributed datasets on the cluster, applies parallel operations to the cluster nodes, and then collects the results of the operations.

Driver programs access Apache Spark through a ***SparkSession*** object. The nodes read and write data from and to the file system and cache transformed data in-memory as ***Resilient Distributed Datasets*** (RDDs). The SparkContext is responsible for converting an application to a ***directed acyclic graph*** (DAG). The graph consists of individual tasks that get executed within an executor process on the nodes. Each application gets its own executor processes, which stays up for the duration of the whole application and run tasks in multiple threads.

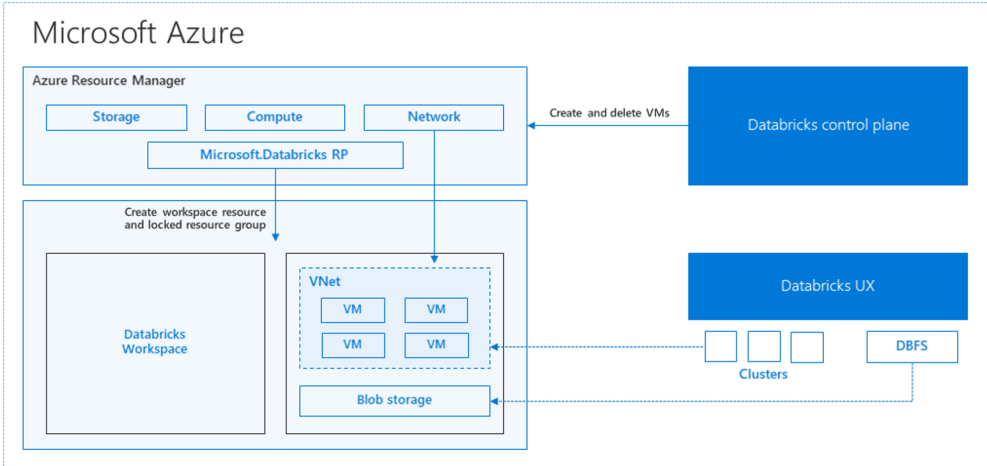
✅ How Azure manages cluster resources

Microsoft Azure manages the cluster, and auto-scales it as needed based on your usage and the setting used when configuring the cluster. Spark parallelizes jobs at two levels:

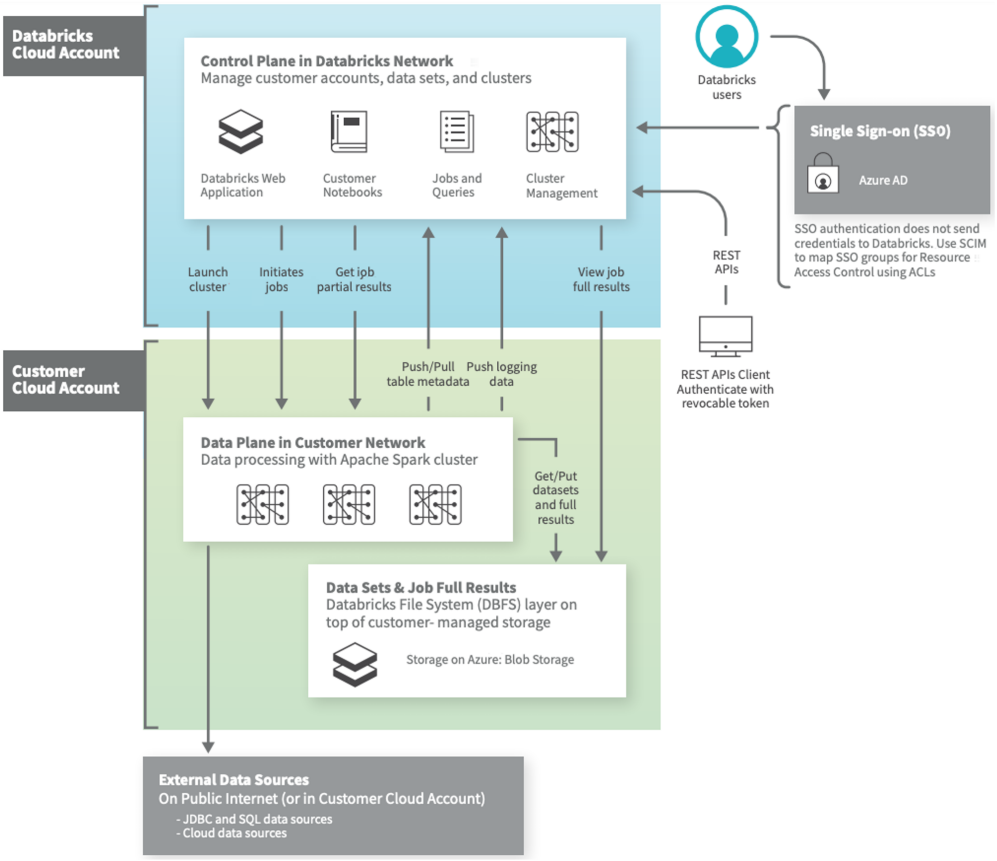
- The first level of parallelization is the ***executor*** - a Java virtual machine (JVM) running on a worker node, typically, one instance per node.
- The second level of parallelization is the ***slot*** - the number of which is determined by the number of cores and CPUs of each node.
- Each executor has multiple slots to which parallelized tasks can be assigned.



When you create an **Azure Databricks workspace**, a resource group is created that contains the driver and worker VMs for your clusters, along with other required resources, including a virtual network, a security group, and a storage account. All metadata for your cluster, such as scheduled jobs, is stored in an Azure Database with geo-replication for fault tolerance. Internally, Azure Kubernetes Service (AKS) is used to run the Azure Databricks control plane and data planes via containers.



Mounting file-based storage to DBFS using Service Principal allows seamless access to data from the storage account without requiring credentials after the first time



✅ Transformations usually performed on a dataset

Basic Transformations

Normalizing values
Missing/Null data
De-duplication
Pivoting Data frames

Advanced Transformations

User Defined functions
Joins and lookup tables
Multiple databases

```
# Mount a data lake dbutils.fs.mount( source = "abfss://<file-system-name>@<storage-account-name>.dfs.core.windows.net/", mount_point = "/mnt/<mount-name>", extra_configs = {config_key:key_name}) # Load a dataframe %%pyspark df = spark.read.load('/data/products.csv', # or 'abfss://container@store.dfs.core.windows.net/data/products.csv' format='csv', header=True ) display(df.limit(10)) # Specify a schema for a dataframe to be loaded from pyspark.sql.types import * from pyspark.sql.functions import * productSchema = StructType([ StructField("ProductID", IntegerType()), StructField("ProductName", StringType()), StructField("Category", StringType()), StructField("ListPrice", FloatType()) ]) df = spark.read.load('/data/product-data.csv', format='csv', schema=productSchema, header=False) display(df.limit(10)) #Filtering pricelist_df = df["ProductID", "ListPrice"] bikes_df = df["ProductName", "ListPrice"].where((df["Category"]=="Mountain Bikes") | (df["Category"]=="Road Bikes")) #Grouping counts_df = df.select("ProductID", "Category").groupBy("Category").count() # We can use the %%sql magic to run SQL code that queries objects in the catalog %%sql SELECT Category, COUNT(ProductID) AS ProductCount FROM products GROUP BY Category ORDER BY Category # PySpark code uses a SQL query to return data bikes_df = spark.sql("SELECT ProductID, ProductName, ListPrice \ FROM products \ WHERE Category IN ('Mountain Bikes', 'Road Bikes')") # Get the data as a Pandas dataframe data = spark.sql("SELECT Category, COUNT(ProductID) AS ProductCount \ FROM products \ GROUP BY Category \ ORDER BY Category").toPandas()
```

✅ Delta lake

Delta Lake is an **open-source storage layer** for Spark that enables relational database capabilities for batch and streaming data. By using Delta Lake, you can implement a *data lakehouse* architecture in Spark to support SQL based data manipulation semantics with support for transactions and schema enforcement. The result is an analytical data store that offers many of the advantages of a relational database system with the flexibility of data file stored in a data lake.

The benefits of using Delta Lake in Azure Databricks include:

- **Relational tables that support querying and data modification**. With Delta Lake, you can store data in tables that support *CRUD* (create, read, update, and delete) operations. In other words, you can *select*, *insert*, *update*, and *delete* rows of data in the same way you would in a relational database system.
- **Support for *ACID* transactions**. Relational databases are designed to support transactional data modifications that provide *atomicity* (transactions complete as a single unit of work), *consistency* (transactions leave the database in a consistent state), *isolation* (in-process transactions can't interfere with one another), and *durability* (when a transaction completes, the changes it made are persisted). Delta Lake brings this same transactional support to Spark by implementing a transaction log and enforcing serializable isolation for concurrent operations.
- **Data versioning and *time travel***. Because all transactions are logged in the transaction log, you can track multiple versions of each table row, and even use the *time travel* feature to retrieve a previous version of a row in a query.

- **Support for batch and streaming data**. While most relational databases include tables that store static data, Spark includes native support for streaming data through the Spark Structured Streaming API. Delta Lake tables can be used as both *sinks* (destinations) and *sources* for streaming data.
- **Standard formats and interoperability**. The underlying data for Delta Lake tables is stored in Parquet format, which is commonly used in data lake ingestion pipelines. Additionally, you can use the serverless SQL pool in Azure Synapse Analytics to query Delta Lake tables in SQL.

```
# Load a file into a dataframe df = spark.read.load('/data/mydata.csv', forma
t='csv', header=True) # Save the dataframe as a delta table delta_table_path
= "/delta/mydata" df.write.format("delta").save(delta_table_path) # Add new r
ows new_rows_df.write.format("delta").mode("append").save(delta_table_path)
```

After saving the delta table, the path location you specified includes **parquet files for the data** (regardless of the format of the source file you loaded into the dataframe) and a **_delta_log** folder containing the transaction log for the table.

Note: The transaction log records all data modifications to the table. By logging each modification, transactional consistency can be enforced and versioning information for the table can be retained.

```
# To make modifications to a Delta Lake table, you can use the DeltaTable obj
ect in the Delta Lake API, which supports update, delete, and merge operation
s. For example, you could use the following code to update the price column f
or all rows with a category column value of "Accessories" from delta.tables i
mport * from pyspark.sql.functions import * # Create a deltaTable object delt
aTable = DeltaTable.forPath(spark, delta_table_path) # Update the table (redu
ce price of accessories by 10%) deltaTable.update( condition = "Category ==
'Accessories'", set = { "Price": "Price * 0.9" })
```

Query the previous version of a table

Delta Lake tables support versioning through the transaction log. The transaction log records modifications made to the table, noting the timestamp and version number for each transaction. You can use this logged version data to view previous versions of the table - a feature known as *time travel*.

You can retrieve data from a specific version of a Delta Lake table by reading the data from the delta table location into a dataframe

```
df = spark.read.format("delta").option("versionAsOf", 0).load(delta_table_pat
h) # OR df = spark.read.format("delta").option("timestampAsOf", '2022-01-0
1').load(delta_table_path)
```

Query catalog tables

You can also define Delta Lake tables as catalog tables in the Hive metastore for your Spark cluster, and work with them using SQL.

Tables in a Spark catalog, including Delta Lake tables, can be *managed* or *external*; and it's important to understand the distinction between these kinds of tables.

- A ***managed*** table is defined without a specified location, and the data files are stored within the storage used by the metastore. Dropping the table not only removes its metadata from the catalog but also deletes the folder in which its data files are stored.
- An ***external*** table is defined for a custom file location, where the data for the table is stored. The metadata for the table is defined in the Spark catalog. Dropping the table deletes the metadata from the catalog, but doesn't affect the data files.

```
# Creating a catalog table from a dataframe # Save a dataframe as a managed t
able df.write.format("delta").saveAsTable("MyManagedTable") ## specify a path
option to save as an external table df.write.format("delta").option("path",
"/mydata").saveAsTable("MyExternalTable") # Creating a catalog table using SQ
L spark.sql("CREATE TABLE MyExternalTable USING DELTA LOCATION '/mydata'") #
We can use the DeltaTableBuilder API (part of the Delta Lake API) to create a
catalog table from delta.tables import * DeltaTable.create(spark) \ .tableNam
e("default.ManagedProducts") \ .addColumn("Productid", "INT") \ .addColumn("P
roductName", "STRING") \ .addColumn("Category", "STRING") \ .addColumn("Pric
e", "FLOAT") \ .execute()
```

✅ Monitoring

1) Ganglia

- Built-in databricks monitoring service that collects data every 15 min by default. We can access this option by going into the cluster and select **Metrics** from the header.

2) Azure Monitor

- No native support for Databricks so setting this up is cumbersome.
- **Dropwizard** is used to send application metrics of Azure Databricks to Azure Monitor whereas **Log4j** is used to send application logs to Azure Monitor.

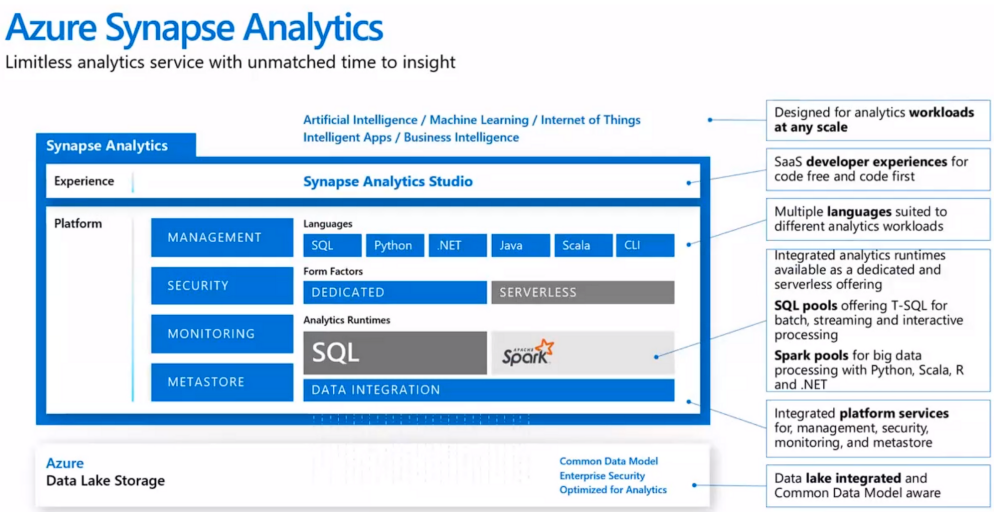
Spark: what to use when and where

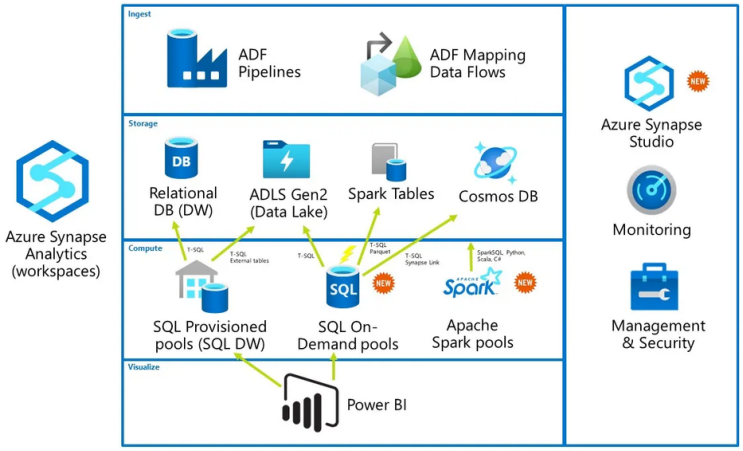
	Apache Spark	HDInsight	Azure Databricks	Synapse Spark
WHAT	Is an Open Source memory optimized system for managing big data workloads	Microsoft implementation of Open Source Spark managed within the realms of Azure	A managed Spark as a Service solution	Embedded Spark capability within Azure Synapse Analytics
WHEN	When you want to benefits of spark for big data processing and/or data science work without the Service Level Agreements of a provider	When you want to benefits of OSS spark with the Service Level Agreement of a provider	Provides end to end data engineering and data science solution and management platform	Enables organizations without existing Spark implementations to fire up a Spark cluster to meet data engineering needs without the overheads of the other Spark platforms listed
WHO	Open Source Professionals	Open Source Professionals wanting SLA's and Microsoft Data Platform experts	Data Engineers and Data Scientists working on big data projects every day	Data Engineers, Data Scientists, Data Platform experts and Data Analysts
WHY	To overcome the limitations of SMP systems imposed on big data workloads	To take advantage of the OSS Big Data Analytics platform with SLA's in place to ensure business continuity	It provides the ability to create and manage an end to end big data/data science project using one platform	It provides the ability to scale efficiently with spark clusters within a one stop shop Data Warehousing platform of Synapse.

Azure provides the Azure Databricks version for customers who love the features of Databricks Spark. It provides HDInsight Spark for customers who prefer OSS technologies, and it also provides Synapse Spark, which is a performance-boosted version of the OSS Spark for those customers who prefer an integrated single-pane experience within Azure Synapse Analytics.

✅ Azure Synapse Analytics - OLAP

> ***Run analytics at a massive scale by using a cloud-based enterprise data warehouse that takes advantage of massively parallel processing (MPP) to run complex queries quickly across petabytes of data.***
> Azure Synapse brings together the best of **SQL** technologies used in enterprise data warehousing, **Spark** technologies used for big data, **Data Explorer** for log and time series analytics, **Pipelines** for data integration and ETL/ELT, and deep integration with other Azure services such as **Power BI**, **CosmosDB**, and **AzureML**.





Synapse Analytics is a unified platform for using ADF, ADLS, Power BI, etc

✅ ASA Top Level Concepts

1) Azure Synapse Pipelines

are cloud-based ETL and data integration service that allows you to create data-driven workflows for orchestrating data movement and transforming data at scale. Azure Synapse uses **Pipelines** (the same Data Integration engine as Azure Data Factory), to create rich at-scale ETL pipelines.

2) Azure Synapse SQL

Synapse SQL is a distributed query system for T-SQL that enables you to implement data warehouse solutions or perform data virtualization. Azure Synapse SQL offers both **dedicated** and **serverless** model of the service (more on this later).

3) Apache Spark for Azure Synapse

Azure Synapse seamlessly integrates Apache Spark for data preparation, data engineering, ETL, and machine learning.

4) Azure Synapse Link

This enables a Hybrid Transactional/Analytical Processing (HTAP) architecture by allowing near-real-time synchronization between operational data in Azure Cosmos DB, Azure SQL Database, SQL Server, and analytical data storage that can be queried in Azure Synapse Analytics.

5) Azure Synapse Data Explorer

This provides an interactive query experience to unlock insights from log and telemetry data using the **Kusto Query Language** (KQL). Data Explorer analytics runtime is optimized for efficient log analytics.

6) Synapse Studio

Synapse Studio provides a single way for enterprises to build solutions, maintain, and secure all in a single user experience using a web-based portal

- Perform key tasks: ingest, explore, prepare, orchestrate, visualize
- Monitor resources, usage, and users across SQL, Spark, and Data Explorer
- Use Role-based access control to simplify access to analytics resources
- Write SQL, Spark, or KQL code and integrate with enterprise CI/CD processes

▸ ✅ **WORKSPACE (SYNAPSE STUDIO)**

▸ ✅ **SYNAPSE SQL (Important)**

▸ ✅ ***DATA EXPLORER (optional)***

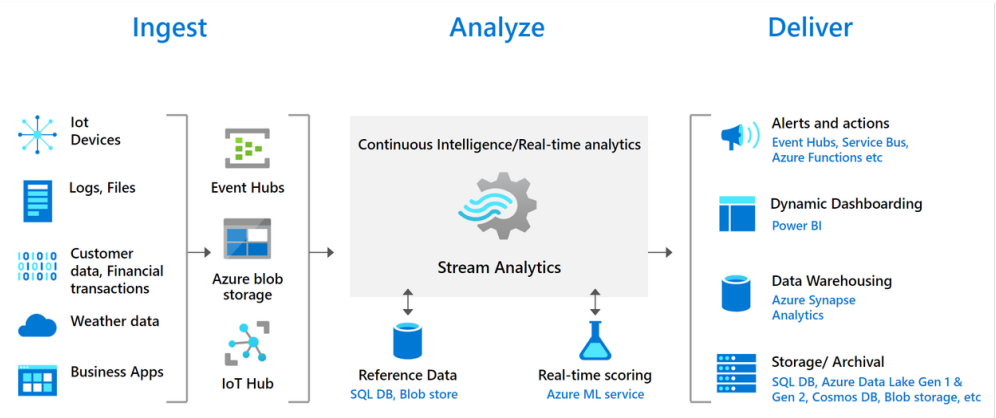
▸ ✅ ***APACHE SPARK (Important)***

▸ ✅ ***SYNAPSE LINK***

▸ ✅ ***PIPELINE AND DATA FLOW (Important)***

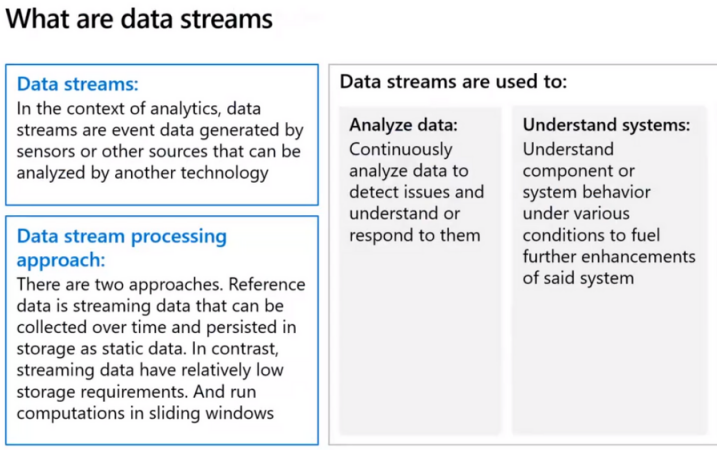
✅ Azure Stream Analytics

> ***Cloud-based stream processing engine (PaaS) solution that can be used to define streaming jobs that ingest data from a streaming source, apply a perpetual query and write the results to output.***



Stream Analytics can route job output to many storage systems such as Azure Blob storage, Azure SQL Database, Azure Data Lake Store, and Azure Cosmos DB. You can also run batch analytics on stream outputs with Azure Synapse Analytics or HDInsight, or you can send the output to another service, like Event Hubs for consumption or Power BI for real-time visualization.

The process of consuming data streams, analyzing them, and deriving actionable insights is called **stream processing**. You can transform streaming data using the **SQL-like Stream Analytics Query Language** to perform temporal and other aggregations against a data stream to gain insights.



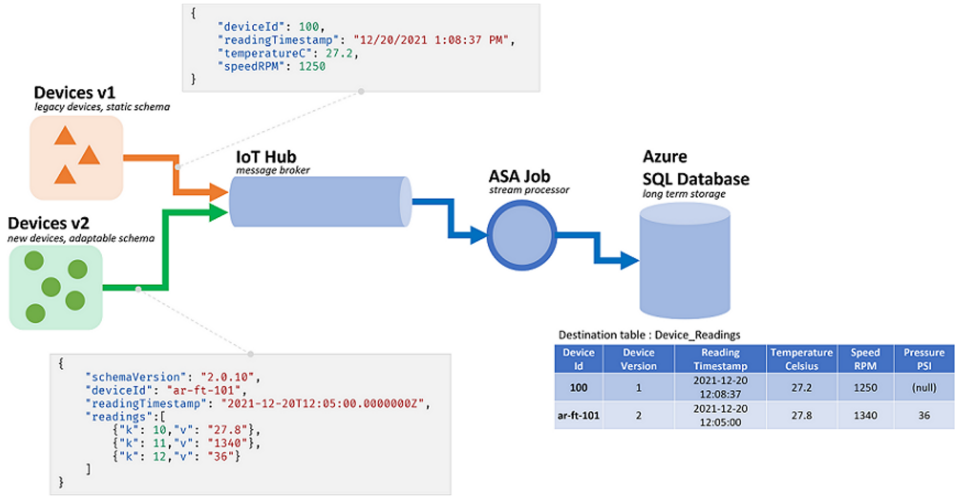
✅ Create a Stream Analytics job (link)

A Stream Analytics job is the fundamental unit in Stream Analytics that allows you to define and run your stream processing logic. A job consists of 3 main components:

1) Input

A job can have one or more inputs to continuously read data from. These streaming input data sources could be Azure Event Hubs, Azure IoT Hub or Azure Storage. Stream Analytics also supports reading static or slow-changing input data (called **reference data**) which is often used to enrich streaming data and perform correlation and lookups.

> **Dynamic schema handling** is a powerful feature, and key to stream processing. Data streams often contain data from multiple sources, with multiple event types, each with a unique schema. To route, filter, and process events on such streams, ASA has to ingest them all whatever their schema.



> But the capabilities offered by dynamic schema handling come with a potential downside. Unexpected events can flow through the main query logic and break it. As an example, we can use **ROUND** on a field of type `NVARCHAR(MAX)`. ASA will implicitly cast it to float to match the signature of `ROUND`. Here we expect, or hope, this field will always contain numeric values. But when we do receive an event with the field set to `"NaN"`, or if the field is entirely missing, then the job may fail.

2) Output

A job can have one or more outputs to continuously write data to.

When you design your Stream Analytics query, refer to the name of the output by using the INTO clause. You can use a single output per job, or multiple outputs per streaming job (if you need them) by adding multiple INTO clauses to the query.

> Stream Analytics supports **partitions** for all outputs except for Power BI. Additionally, for more advanced tuning of the partitions, the number of output writers can be controlled using an `INTO <partition count>` clause in your query, which can be helpful in achieving a desired job topology.

```
WITH Step1 AS ( SELECT * FROM input PARTITION BY DeviceId INTO 10 ) SELEC
T * INTO [output] FROM Step1 PARTITION BY DeviceId
```

3) Query

The rich SQL like language support allows you to tackle scenarios such as parsing complex JSON, filtering values, computing aggregates, performing joins, and even more advanced use cases such as geospatial analytics and anomaly detection. We can also extend this SQL language with JavaScript or C# user-defined functions (UDF) and JavaScript user-defined-aggregates (UDA).

✅ Create a Stream Analytics cluster

A Stream Analytics cluster is a single-tenant deployment that can be used for complex and demanding streaming use cases. You can run multiple Stream Analytics jobs on a Stream Analytics cluster.

By default, Stream Analytics jobs run in the Standard multi-tenant environment which forms the **Standard SKU**. Stream Analytics also provides a **Dedicated SKU** where you can provision an entire Stream Analytics cluster that belongs to you.

Streaming Unit Capacity are available from **36 SUs through 396 SUs (36, 72, 108...)**. We need to determine the size of the cluster by estimating how many Stream Analytics job we plan to run and the total SUs the job will require. We can scale up or down as required. ***(36 SUs mean approximately 36 MB/second throughput with millisecond latency).***

✅ Understand and Adjust Streaming Units (SUs)

> ***Streaming Units (SUs) represent the computing resources that are allocated to execute a Stream Analytics job. The higher the number of SUs, the more CPU and memory resources are allocated for your job.***

To achieve low latency stream processing, Azure Stream Analytics jobs perform all **processing in-memory**. When running out of memory, the streaming job fails. The SU % utilization metric describes the memory consumption of your workload.

One of the unique capability of Azure Stream Analytics job is to perform stateful processing, such as windowed aggregates, temporal joins, and temporal analytic functions. Each of these operators keeps state information.

The temporal window concept appears in several Stream Analytics query elements. The following factors influence the memory used (part of streaming units metric)

1. **Windowed aggregates:** `GROUP BY` of Tumbling, Hopping, and Sliding windows

   The memory consumed (state size) for a windowed aggregate isn't always directly proportional to the window size. Instead, the memory consumed is proportional to the cardinality of the data, or the number of groups in each time window. We can use GROUP BY clause to reduce cardinality.

   ```
   SELECT count(*) FROM input PARTITION BY PartitionId GROUP BY PartitionId,
   clusterid, TumblingWindow (minutes, 5)
   ```

2. **Temporal joins:** `JOIN` with `DATEDIFF` function

   The memory consumed (state size) of a temporal join is proportional to the number of events in the temporal wiggle room of the join, which is event input rate multiplied by the wiggle room size. For reading more about this, click here

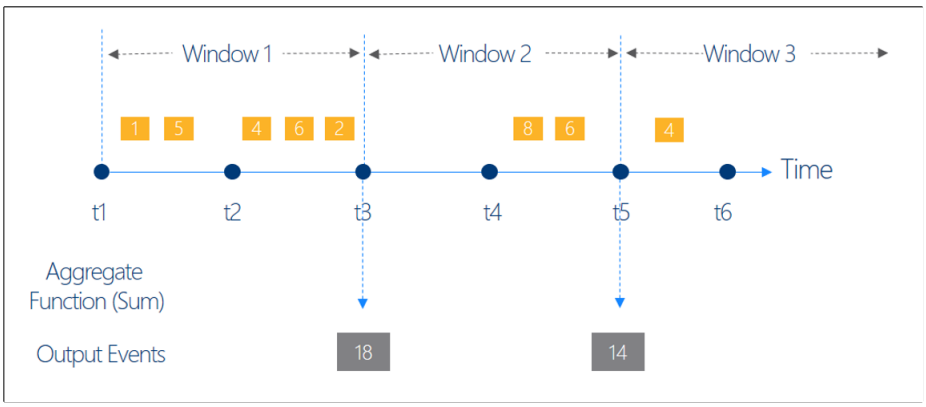
3. **Temporal analytic functions:** `ISFIRST`, `TOPONE`, `LAST`, and `LAG` with LIMIT DURATION

   The memory consumed (state size) of a temporal analytic function is proportional to the event rate multiply by the duration. The memory consumed by analytic functions isn't proportional to the window size, but rather partition count in each time window.

✅ Windowing Functions

> ***Windowing functions are operations performed against the data contained within a temporal or time-boxed window. A window contains event data along a timeline. Using windowing provides a way to aggregate events over various time intervals depending on specific window definitions.***

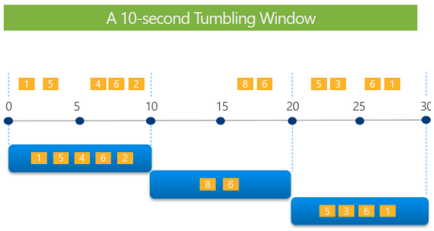
Stream Analytics has native support for windowing functions. There are five kinds of temporal windows to choose from: **Tumbling**, **Hopping**, **Sliding**, **Session**, and **Snapshot** windows. You use the window functions in the **GROUP BY** clause of the query syntax in your Stream Analytics jobs. You can also aggregate events over multiple windows using the **Windows**() function.



1) Tumbling window

Tumbling window functions are used to **segment a data stream into distinct time segments** and perform a function against them, such as in the example below. The key differentiators of a Tumbling window are that they repeat, do not overlap, and an event cannot belong to more than one tumbling window.

Tell me the count of Tweets per time zone every 10 secor




```
SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
```

By default, windows are inclusive of the end of the window and exclusive of the beginning. However, you can use the `Offset` parameter to change this behavior.

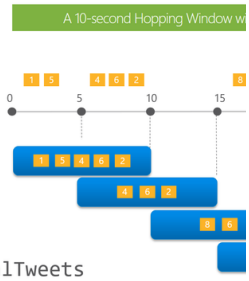
▶ **Complex Code Example**

2) Hopping window

Hopping window functions **hop forward in time by a fixed period**. It may be easy to think of them as Tumbling windows that can overlap. Events can belong to more than one Hopping window result set.

The `windowsize` is 10 seconds, and the `hopsize` is 5 seconds

Every 5 seconds give me the count of Tweets over the last 10 seconds



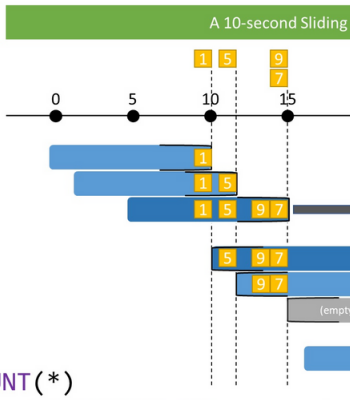

```
SELECT Topic, COUNT(*) AS TotalTweets
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, HoppingWindow(second, 10 , 5)
```

3) Sliding window

Sliding windows, unlike Tumbling or Hopping windows, **output events only for points in time when the content of the window actually changes**. In other words, when an event enters or exits the window. **So, every window has at least one event.**

Alert me whenever a topic is mentioned more than 3 times in under 10 seconds

Note:
- all tweets on the diagram belong to the same topic

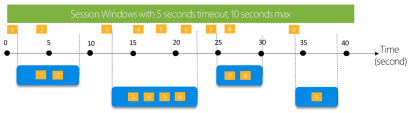



```
SELECT Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, SlidingWindow(second, 10)
HAVING COUNT(*) >= 3
```

4) Session window

Session window **cluster together events that arrive at similar times**, filtering out periods of time where there is no data.

Tell me the count of Tweets that occur within 5 seconds of each other




```
SELECT System.Timestamp() as WindowEnd, Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, SessionWindow(second, 5, 10)
```

The following query measures user session length by creating a `SessionWindow` over clickstream data with a `timeoutsize` of 5 seconds and a `maximumdurationsize` of 10 seconds

A session window begins when the first event occurs. If another event occurs within the specified timeout from the last ingested event, then the window extends to include the new event. Otherwise, if no events occur within the timeout, then the window is closed at the timeout.

If events keep occurring within the specified timeout, the session window will keep extending until the maximum duration is reached. The maximum duration checking intervals are set to be the same size as the specified max duration. For example, if the max duration is 10, then the checks on if the window exceeds the maximum duration will happen at t = 0, 10, 20, 30, etc.

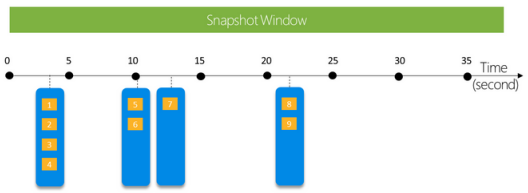
When a `partition` key is provided, the events are grouped together by the key and the session window is applied to each group independently. This partitioning is useful for cases where you need different session windows for different users or devices.

▶ **Code Example with `partition by`**

5) Snapshot window

Snapshot windows group events that have the same timestamp. Unlike other windowing types, which require a specific window function, you can apply a snapshot window by adding `System.Timestamp()` to the GROUP BY clause.

Give me the count of tweets with the same topic type that occur at exactly the same time




```
SELECT System.Timestamp() as WindowEnd, Topic, COUNT(*)
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY Topic, System.Timestamp()
```

✅ Monitoring Performance + Metrics