

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.



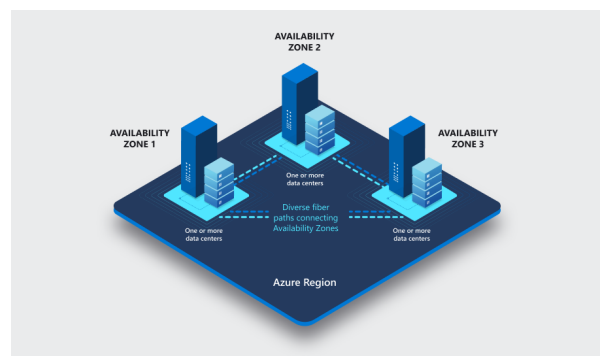
Services such as **Azure Synapse Analytics, Azure Databricks, and Azure Data Factory** have native functionality that takes advantage of Parquet file formats.



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

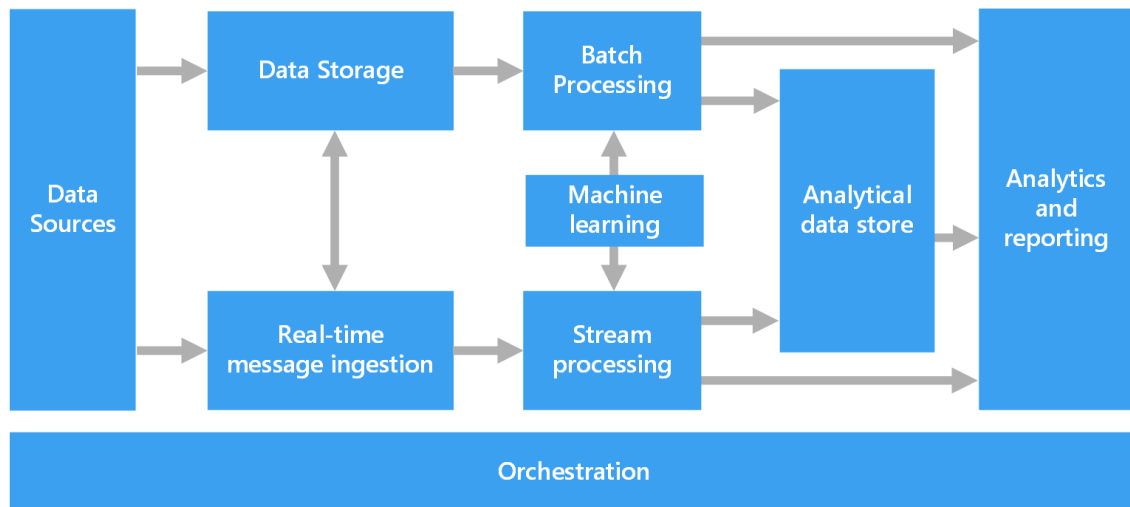
✓ OLTP vs OLAP

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?

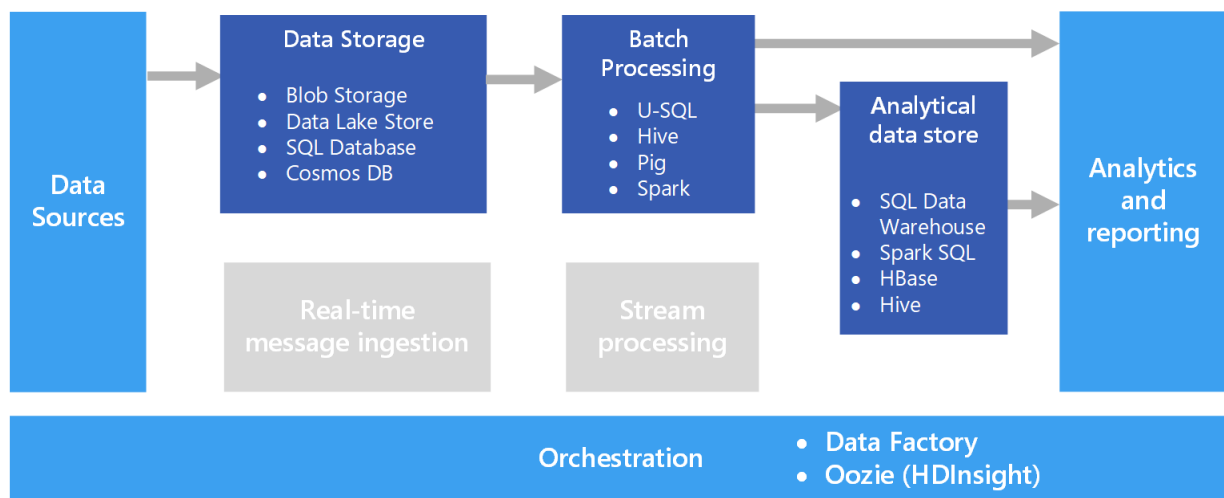
AREA	DATA LAKE	DATA WAREHOUSE
Data Store 	It can capture and retain unstructured, semi-structured, and structured data in its raw format. A Data Lake stores all types of data, irrespective of the source and structure.	It can capture and retain only structured data. A Data Warehouse stores data in quantitative 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 as 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](#))**



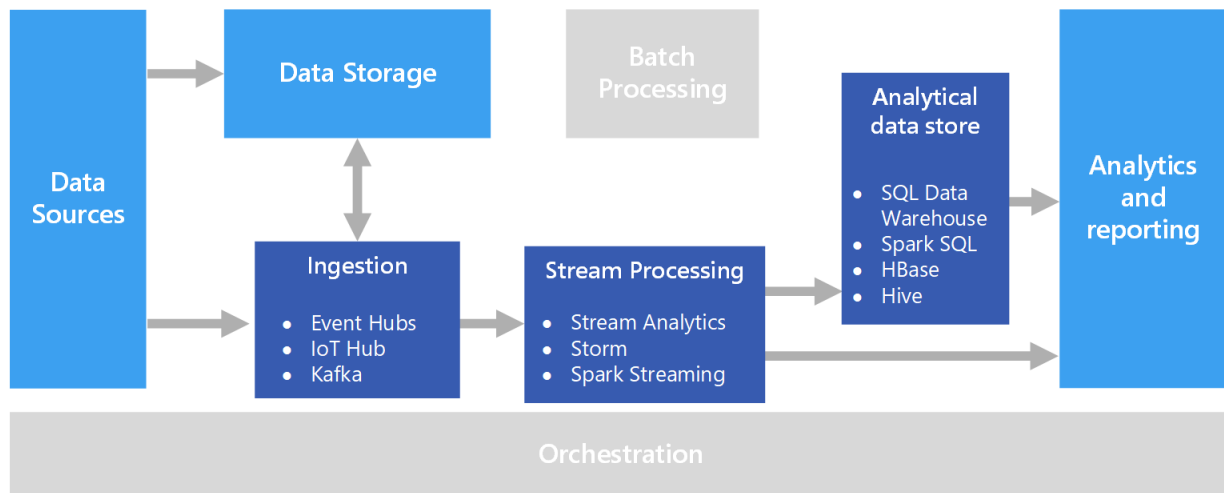
✓ Batch processing

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.



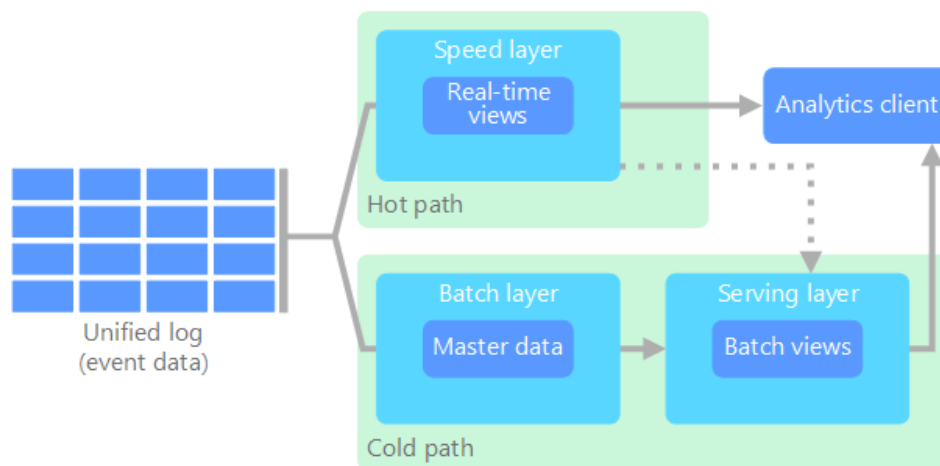
✓ Stream processing

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:

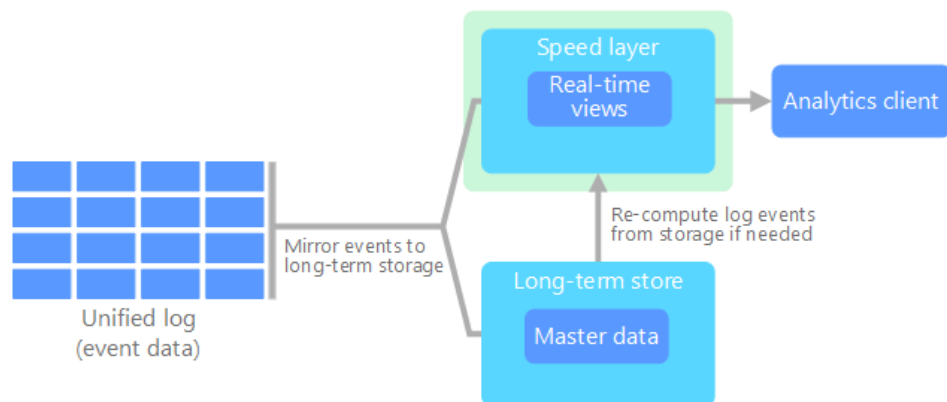
- A **batch layer** (cold/slow path) stores all of the incoming data in its raw form and performs batch processing on the data. The result of this processing is stored as a **batch view**.
- A **speed layer** (hot/fast path) analyzes data in real time. This layer is designed for low latency, at the expense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

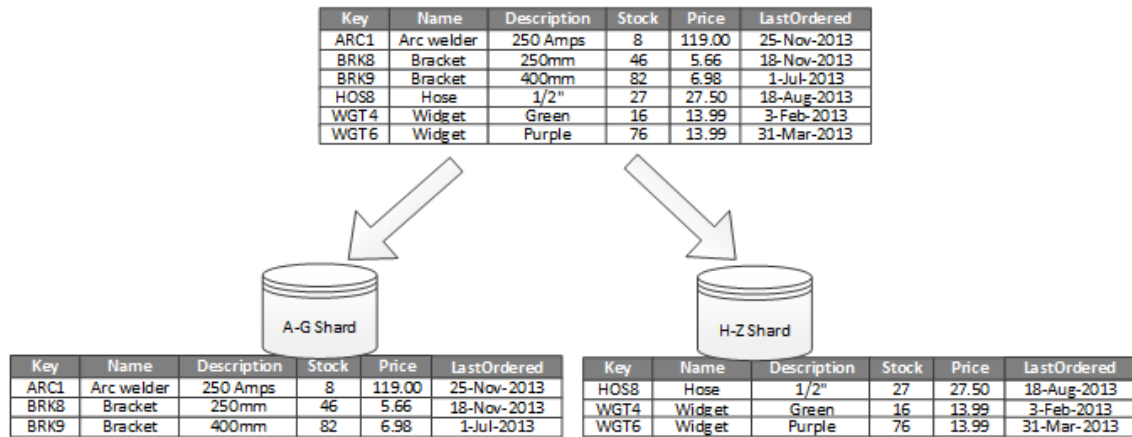
In many large-scale solutions, data is divided into *partitions* that can be managed and accessed separately. Partitioning can improve scalability, reduce contention, and optimize performance. It can also provide a mechanism for dividing data by usage pattern.

There are three typical strategies for partitioning data:

1) Horizontal partitioning (often called *sharding*)

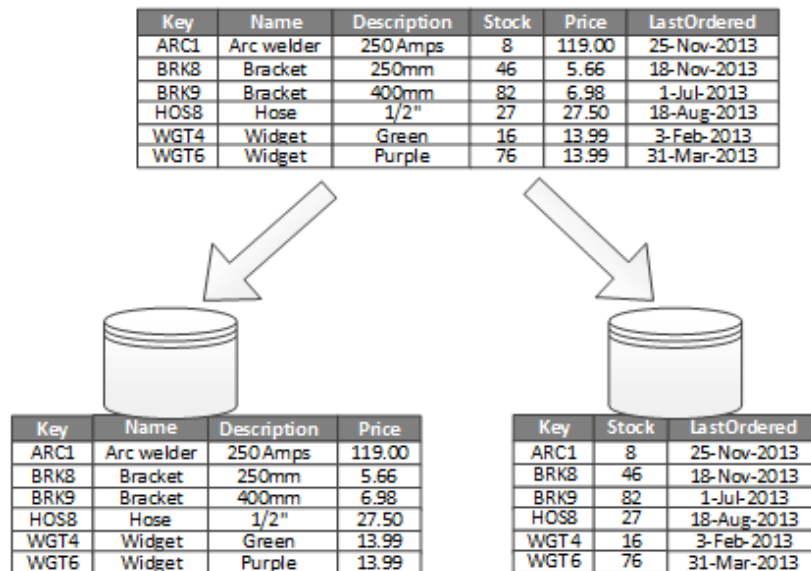
In this strategy, each partition is a separate data store, but all partitions have the same schema. Each partition is known as a *shard* and holds a specific subset of the data, such

as all the orders for a specific set of customers.



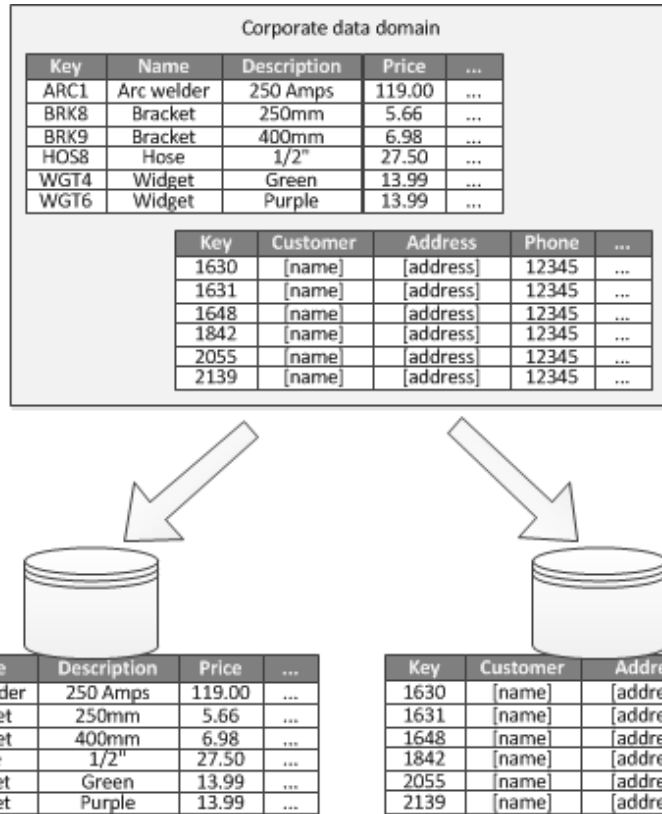
2) Vertical partitioning

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.



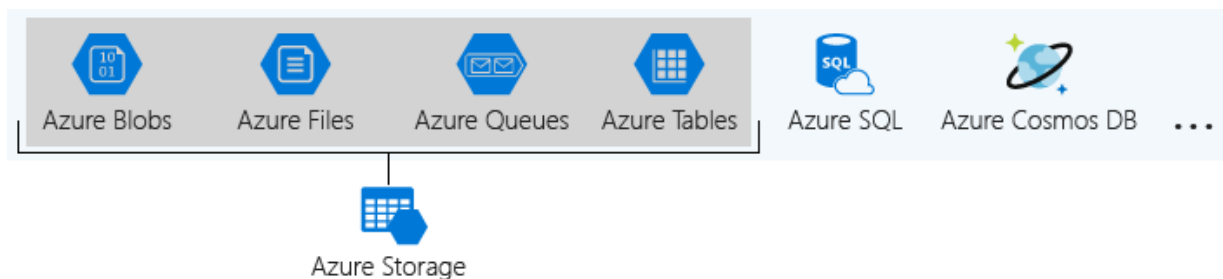
3) Functional partitioning

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.



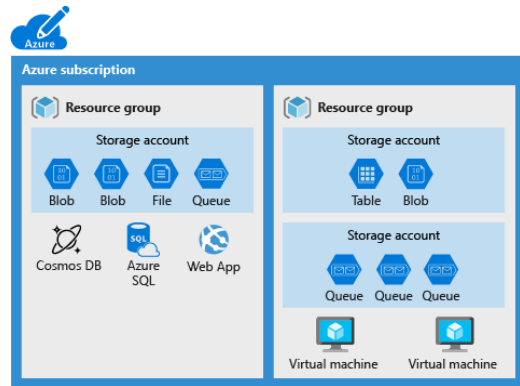
✓ Azure Storage

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).

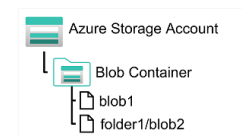
Other Azure data services, such as Azure SQL and Azure Cosmos DB, are managed as independent Azure resources and cannot be included in a storage account.



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.

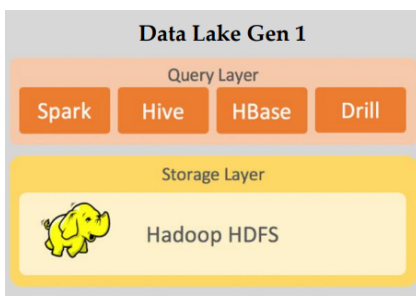
▼ → Hadoop HDFS vs DataLake (optional)

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.

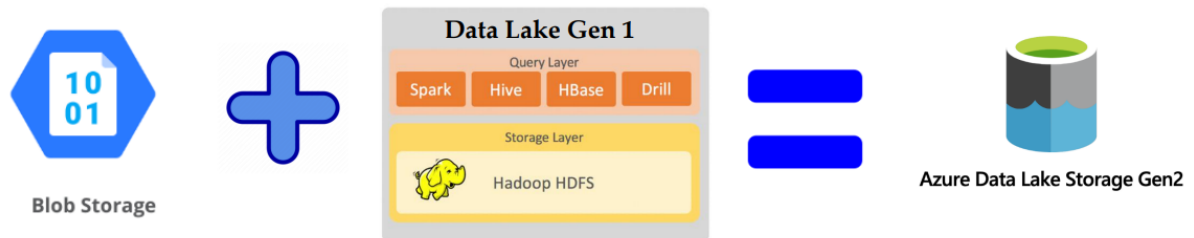


Hadoop already has inbuilt advantages such as a fault-tolerant file system, the ability to run on commodity hardware, etc. Microsoft utilized these advantages by creating Data Lake Gen 1 which is basically Hadoop in the cloud.

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**



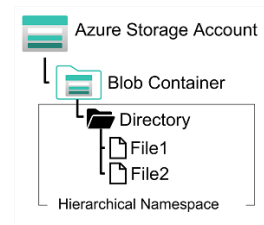
Differences:

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}/
```

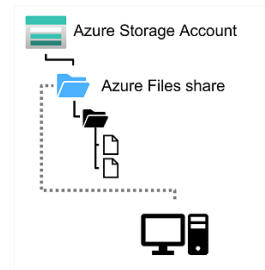


TIP: Avoid putting date folders at the beginning as it makes applying ACLs to every subfolder more tedious.

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.

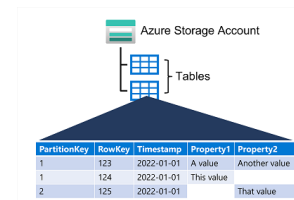
There are two types of queues: **Storage queues** and **Service Bus**.

- 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

Cool access tier:

Lower storage costs, but higher access and transaction costs.

Optimized for data that is infrequently accessed and stored for **at least**

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**

accessed frequently (for example, images for your website).

30 days (for example, invoices for your customers).

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

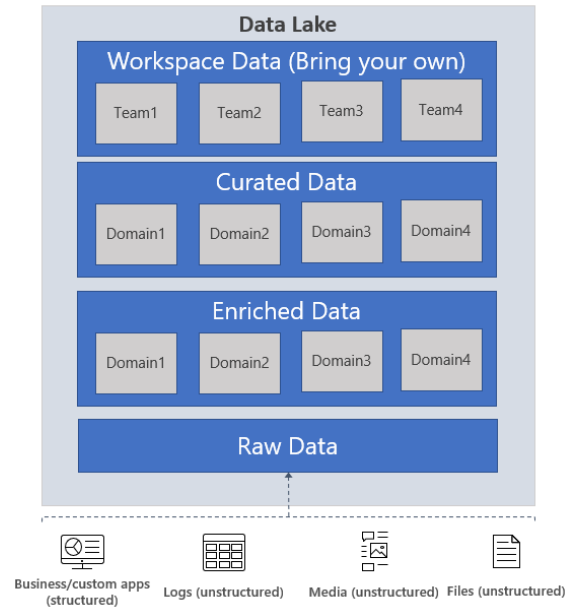
Since ADLS is hierarchical in nature, implementing a folder structure will take care of the partitioning strategy. Try to follow something like

```
{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

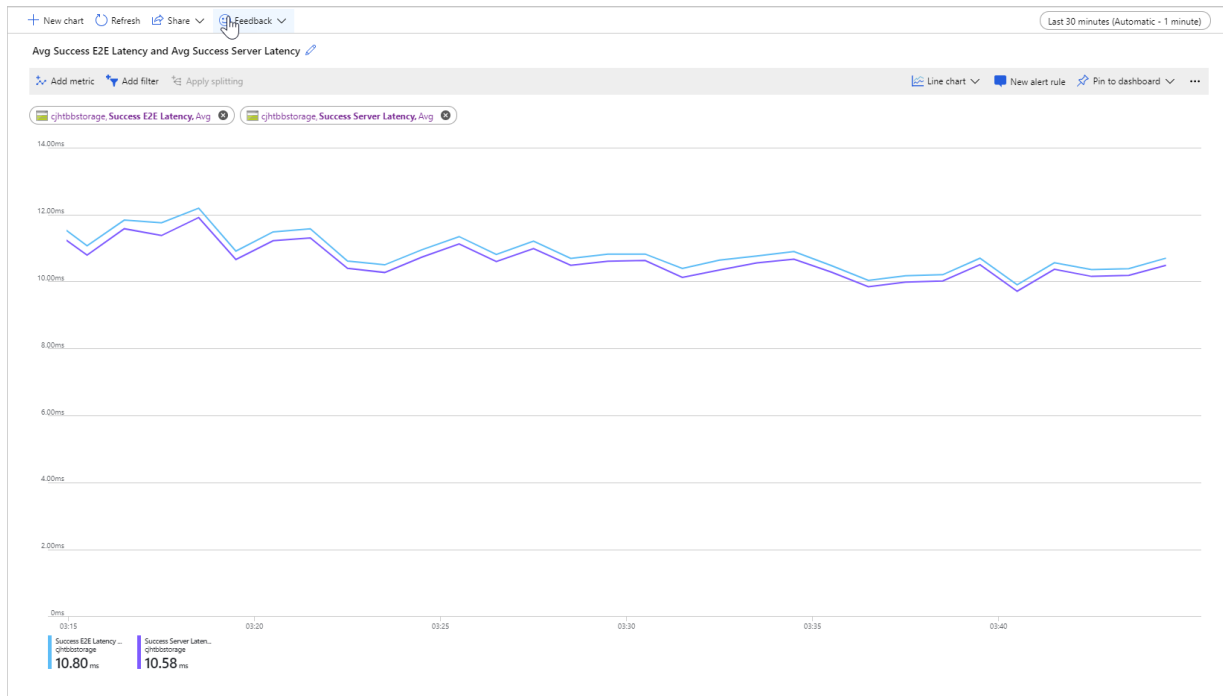
Request rate is measured in Input/output operations per second (IOPS). The request rate is calculated by dividing the time it takes to complete one request by the number of requests per second. E.g. Let us assume that a request from a single thread application with one outstanding read/write operation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

Request Rate = $1\text{sec}/10\text{ms} = 1000\text{ms}/10\text{ms} = 100$ 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:

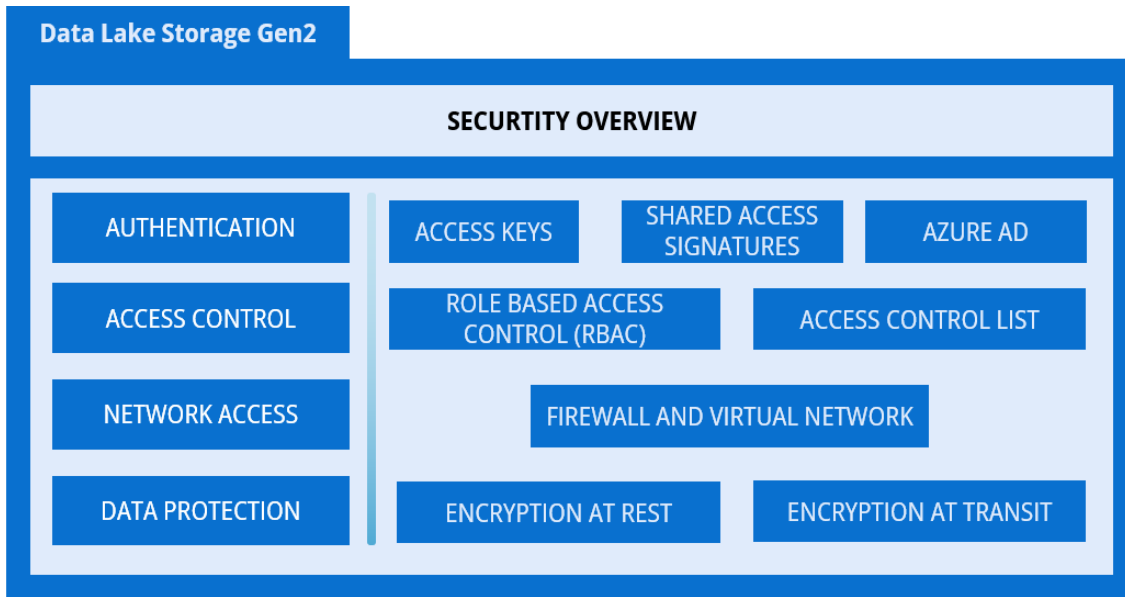
- **End-to-end (E2E) latency** measures the interval from when Azure Storage receives the first packet of the request from a client until Azure Storage receives a client acknowledgment on the last packet of the response.
- **Server latency** measures the interval from when Azure Storage receives the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



✓ Storage Account Security Features

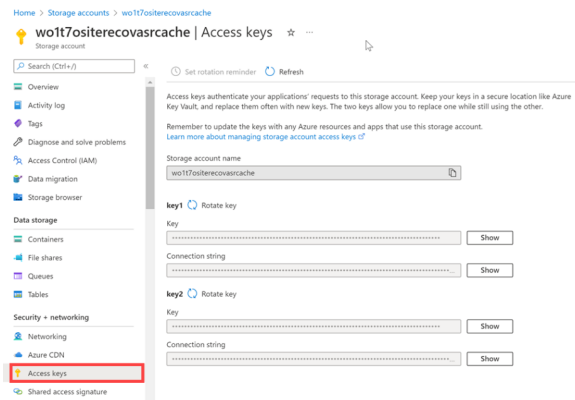
Azure Storage provides a layered security model. We can use this model to secure our storage accounts to a specific set of supported networks. Network rules allow only applications that request data over the specified networks to access our storage account.

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.

Shared access signature

A shared access signature (SAS) is a URI that grants restricted access rights to Azure Storage resources. You can provide a shared access signature to client access signature URI to these clients, you grant them access to a resource for a specified period of time.

An account-level SAS can delegate access to multiple storage services (i.e. blob, file, queue, table). Note that stored access policies are currently not supported.

Learn more

Allowed services Blob File Queue Table

Allowed resource types Service Container Object

Allowed permissions Read Write Delete List Add Create Update Process

Blob versioning permissions Enables deletion of versions

Start and expiry date/time Start 07/27/2021 End 07/28/2021

(UTC+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna

Allowed IP addresses for example, 168.1.5.65 or 168.1.5.65-168.1.5.70

Allowed protocols HTTPS only HTTPS and HTTP

Preferred routing tier Basic (default) Microsoft network routing Internet routing

Some routing options are disabled because the endpoints are not published.

Signing key key1

Generate SAS and connection string



*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.*

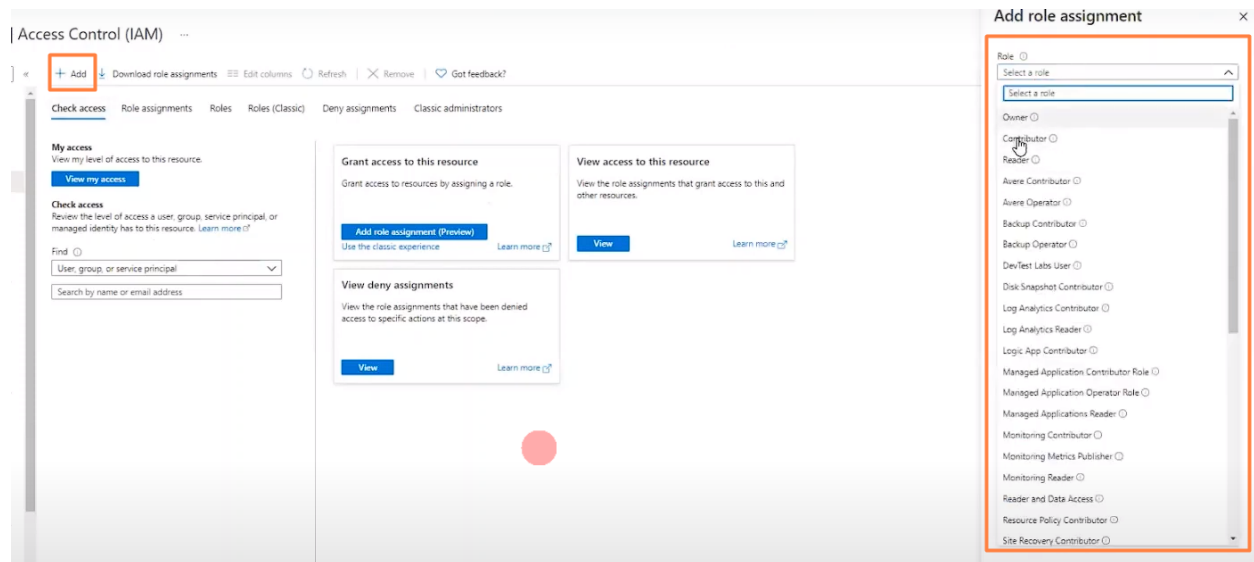
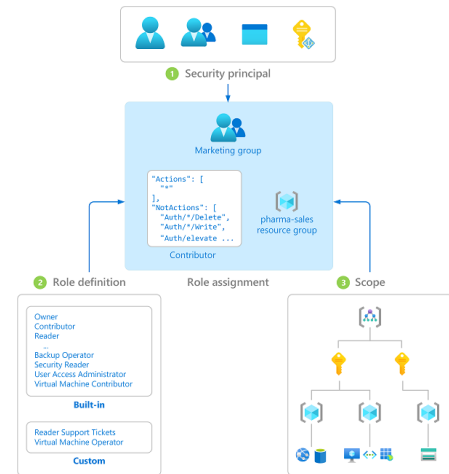
3) Role-based access control (RBAC)

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 Principal:** 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.

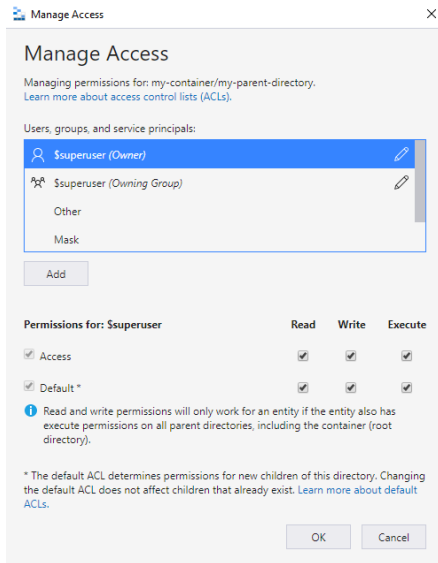
2) **Role Definition:** A role definition is a collection of permissions. It's typically just called a role. A role definition lists the actions that can be performed, such as read, write, and delete. Azure has a huge list of predefined roles, such as Owner, Contributor, and Reader, etc with the right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



1 Security principal



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.

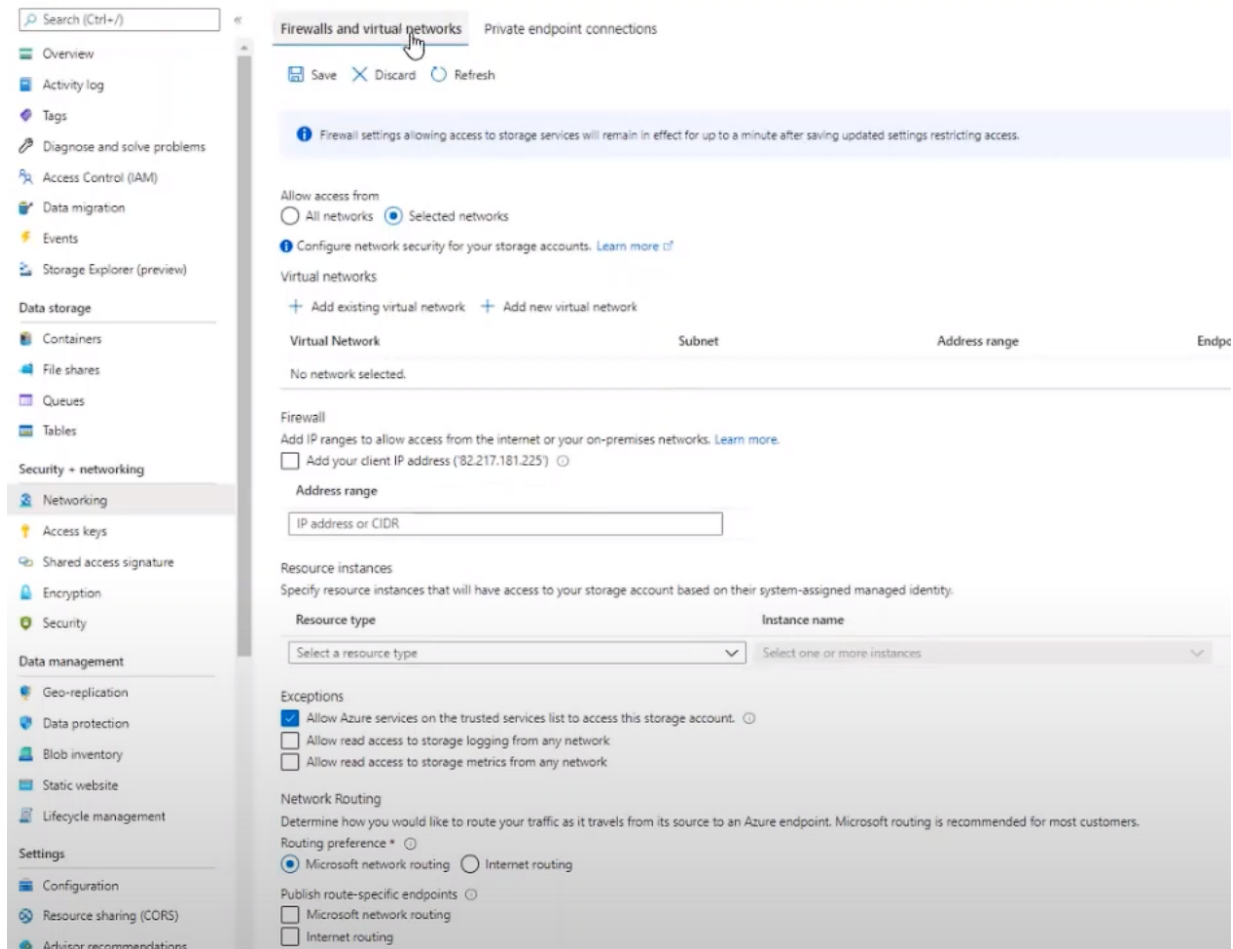


RBAC provides course grain permissions to the data lake or to folders inside it. These are used to allow or deny permissions to the **folder structure** but typically **do not dictate** the ability of the **user** to perform actions against the data.

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

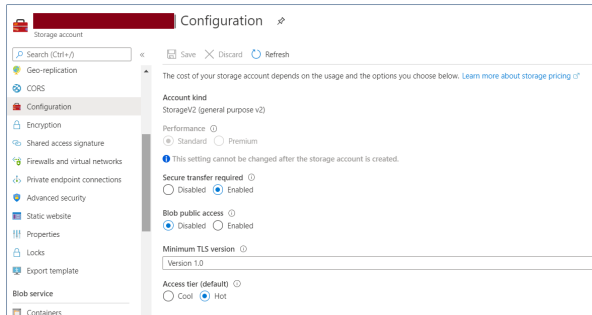
Azure Storage provides a layered security model. Storage accounts having a public endpoint is accessible through the internet. We can also create Private Endpoints for your storage account, which assigns a private IP address from our VNet to the storage account, and secures all traffic between our VNet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

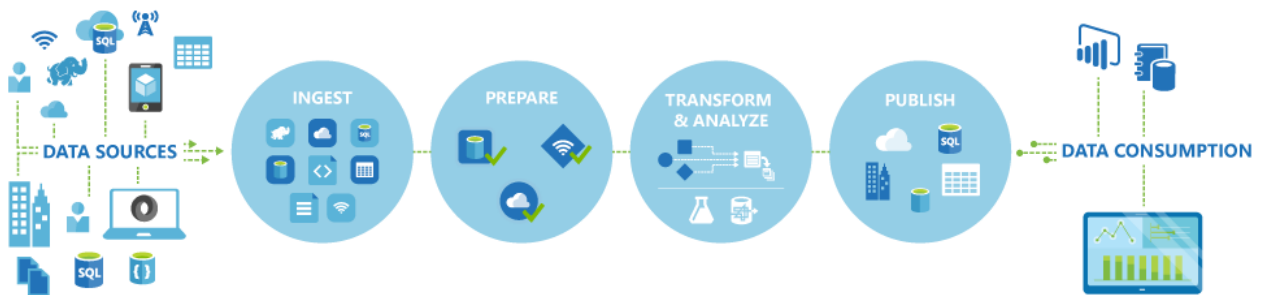
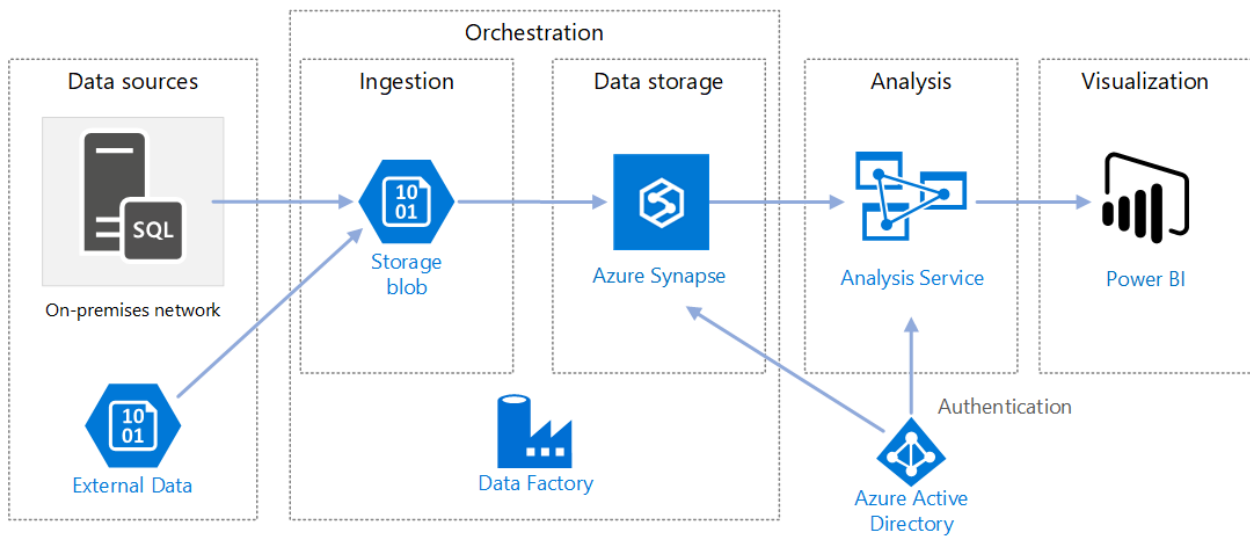
Encryption at rest is **enabled by default** and **can't be disabled**. All data written to Azure Storage is automatically encrypted by **Storage Service Encryption (SSE)** with a **256-bit Advanced Encryption Standard (AES)** cipher.

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

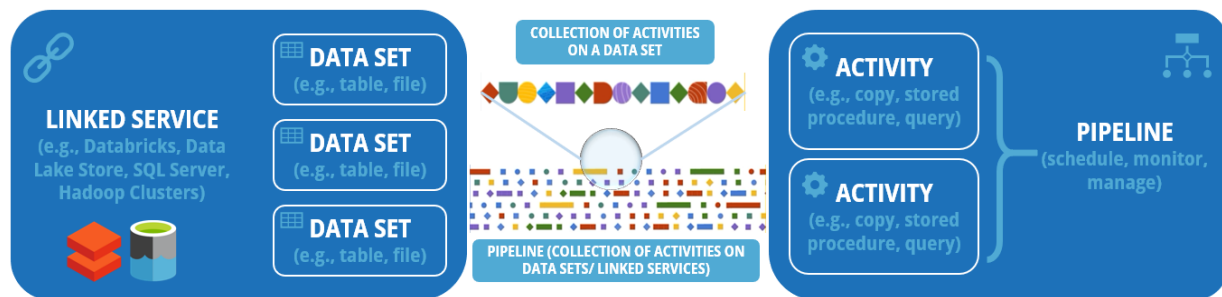
Monitor Phase

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.

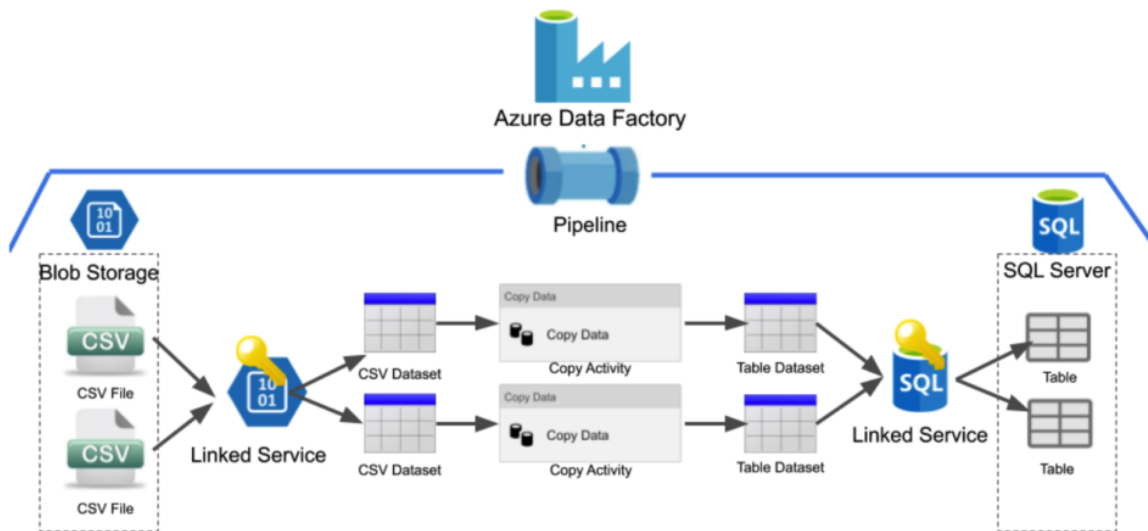
Now, a **dataset** is a named view of data that simply points or references the data you want to use in your **activities** as inputs and outputs. Before you create a dataset, you must create a **linked service** to link your data store to the Data Factory. **Linked services** are like connection strings,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

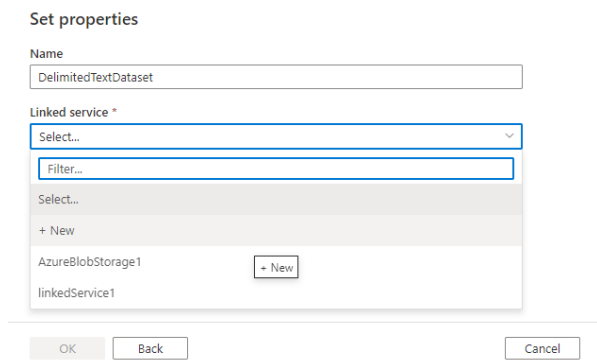
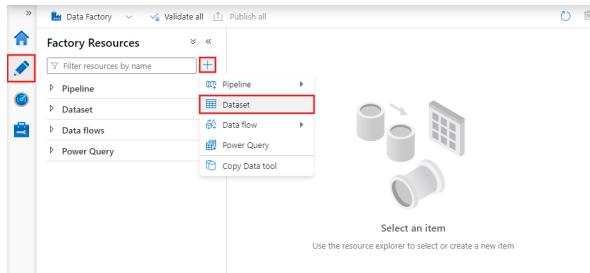


✓ Linked Service

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:

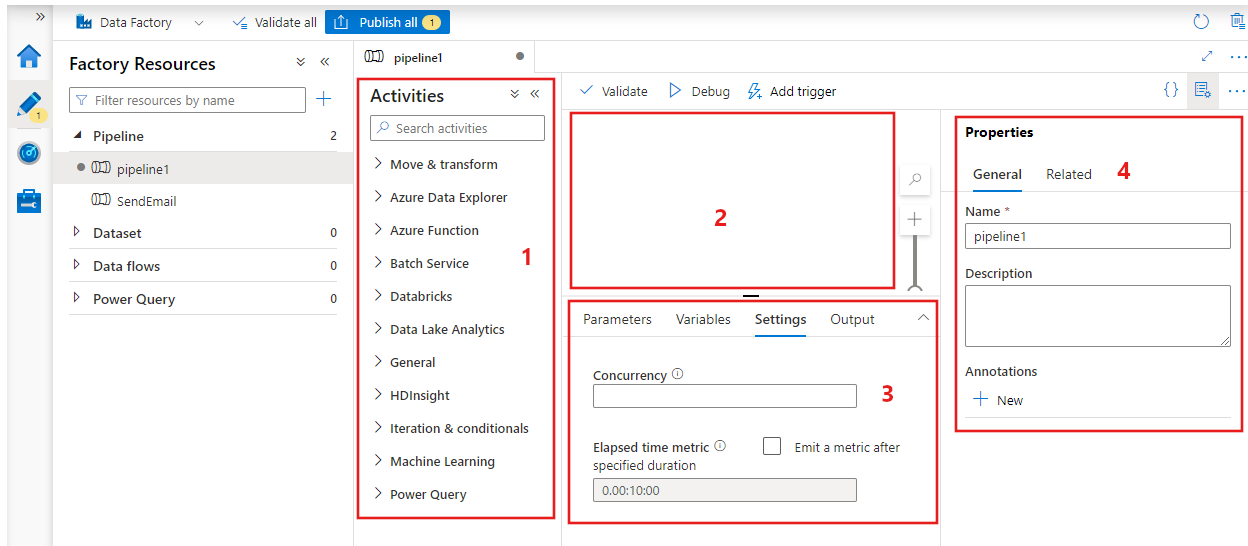
When monitoring data pipelines, you may want to be able to filter and monitor a certain group of activities, such as those of a project or specific department's pipelines. You can achieve these using annotations.

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**.

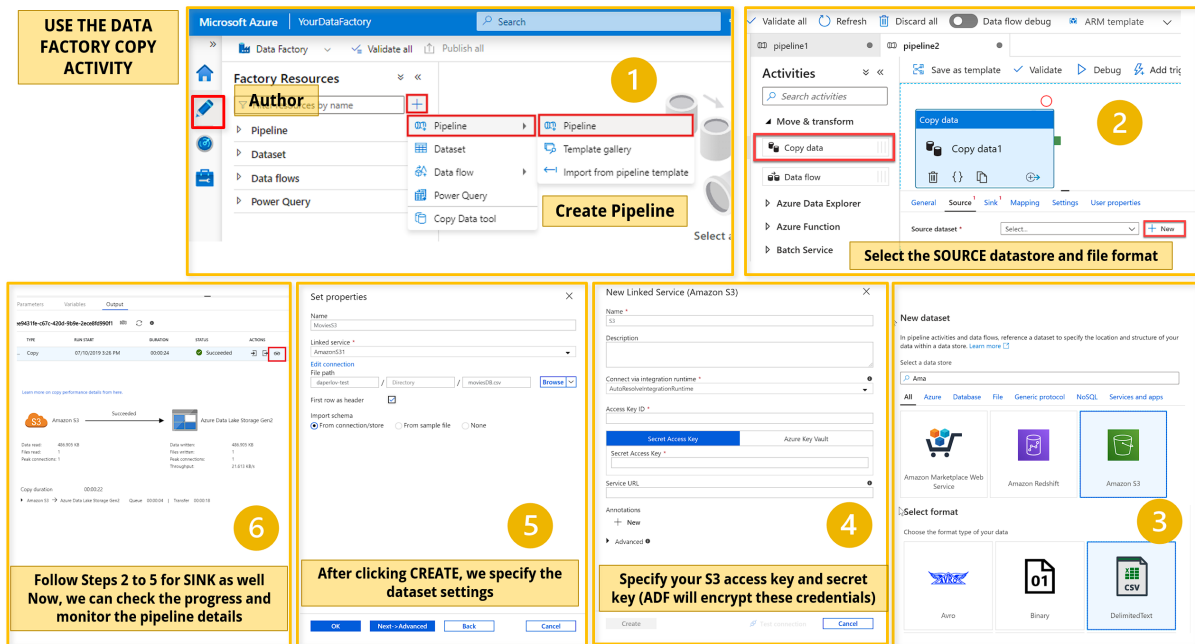


1. All activities that can be used within the pipeline.
2. The pipeline editor canvas, where activities will appear when added to the pipeline.
3. The pipeline configurations pane, including parameters, variables, general settings, and output.
4. The pipeline properties pane, where the pipeline name, 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](#).

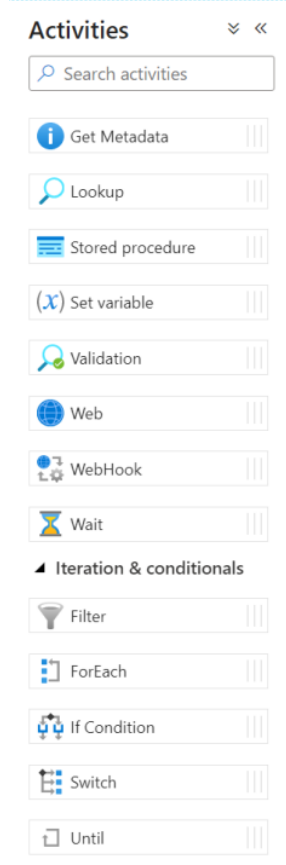
3) Control flow activities:

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 the 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.



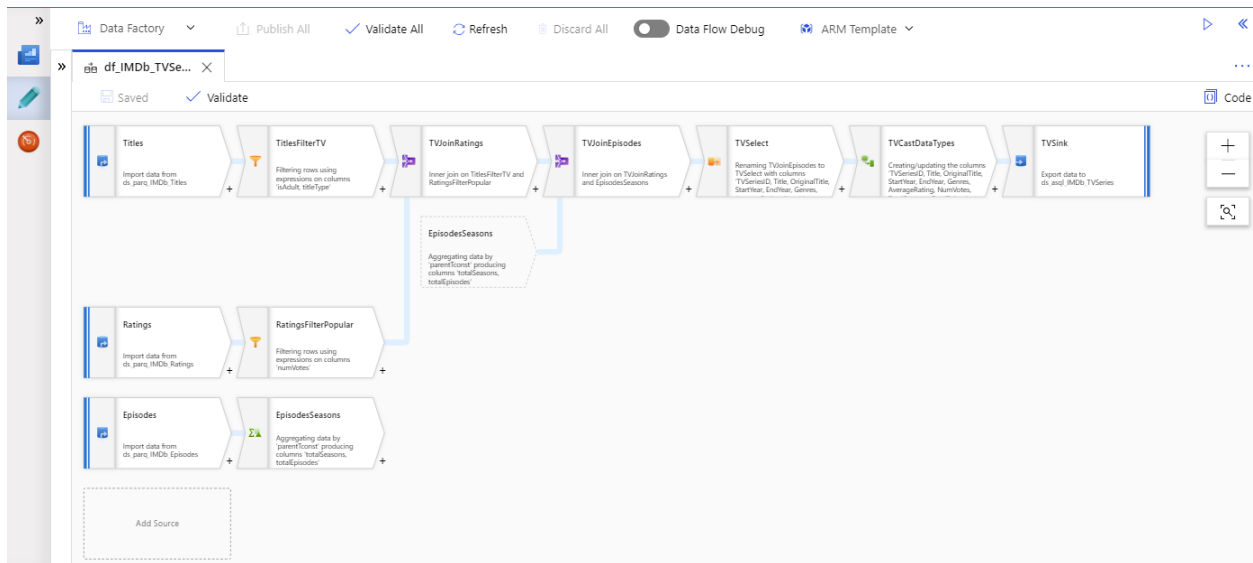
For reading more about these individual activities, click [here](#).

✓ Data Flows

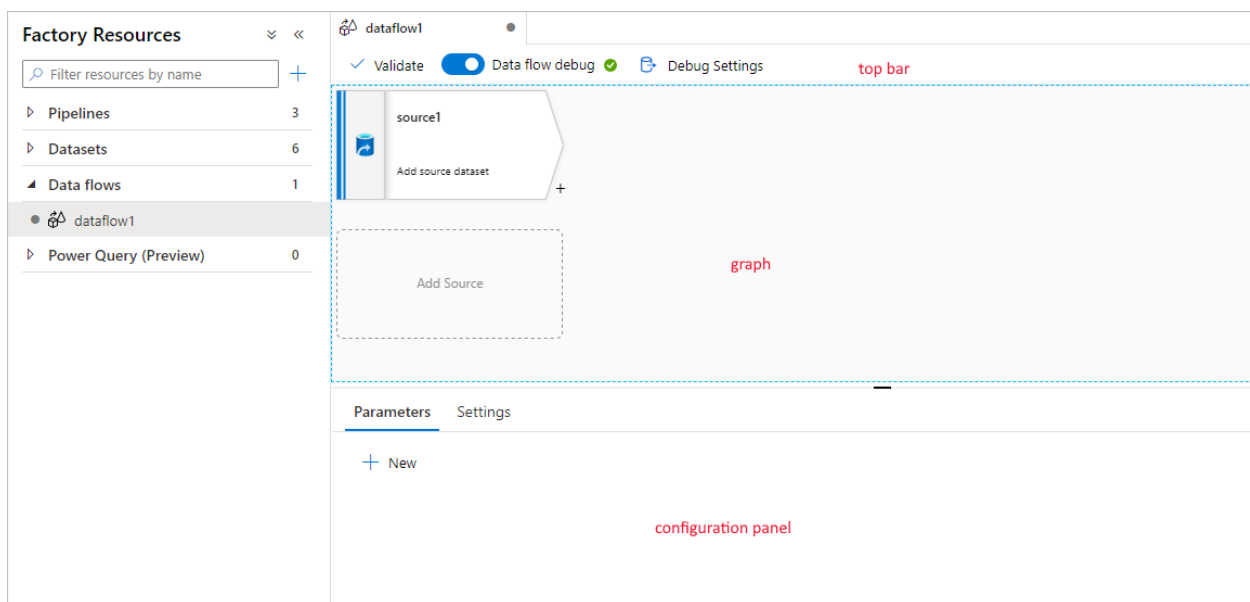
***Data Flows** are used to build code-free transformation data flows/ transformation logic that is executed on **automatically provisioned Apache Spark clusters**. ADF internally handles all the code translation, spark optimization, and execution of the transformation.*

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)

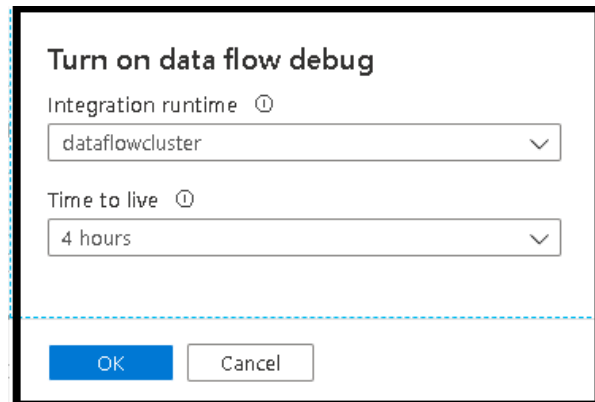


Data flow has a unique authoring canvas designed to make building transformation logic easy. The data flow canvas is separated into three parts: the top bar, the graph, and the configuration panel.



— **Debug mode:** Here you can actually

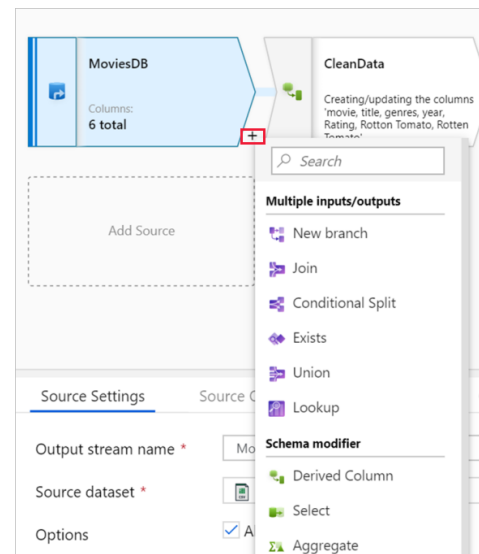
see the results of each transformation. In the debug mode session, the data flow runs interactively on a **Spark cluster**. In the debug mode you will be charged on an hourly basis when the cluster is active. It typically takes 5-7 minutes for the cluster to spin up. With this mode, you are able to build 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.

The screenshot shows the 'Source settings' tab of a configuration pane. It includes the following fields and options:

- Output stream name ***: A text input field containing 'Source' and a 'Learn more' link.
- Source type ***: A dropdown menu set to 'Dataset'.
- Dataset ***: A dropdown menu set to 'ADLSTGen2Input', with 'Test connection', 'Open', and 'New' buttons.
- Options**: Three checkboxes: 'Allow schema drift' (checked), 'Infer drifted column types', and 'Validate schema'.
- Skip line count**: An empty text input field.
- Sampling ***: Radio buttons for 'Enable' and 'Disable' (selected), with an information icon.

2) Optimize

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

The screenshot shows the 'Optimize' tab of a configuration pane. It includes the following fields and options:

- Partition option ***: Radio buttons for 'Use current partitioning', 'Single partition', and 'Set Partitioning' (selected).
- Partition type ***: Five diagrammatic options: 'Round Robin' (highlighted with a blue box), 'Hash', 'Dynamic Range', 'Fixed Range', and 'Key'.
- Number of partitions ***: A text input field containing '20'.

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.

Derived column's settings		Optimize	Inspect	Data Preview	Description
Output schema		Input schema			
Number of columns	New ⁺ 1	Updated [*] 2	Unchanged 4	Total 7	
Order ↓	Column ↓	Type ↓	Updated ↓	Based on ↓	
1	movie	abc string			
2	title	abc string	*	title	
3	genres	abc string			
4	year	121 long	*	year	
5	Rating	abc string			
6	Rotten Tomato	abc string			
7	Rotten Tomato	121 long	+	Rotten Tomato	

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.

movielid	title	genres
1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
2	Jumanji (1995)	Adventure Children Fantasy
3	Grumpier Old Men (1995)	Comedy Romance
4	Waiting to Exhale (1995)	Comedy Drama Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action Crime Thriller
7	Sabrina (1995)	Comedy Romance
8	Tom and Huck (1995)	Adventure Children
9	Sudden Death (1995)	Action

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

Multiple input/output transformations

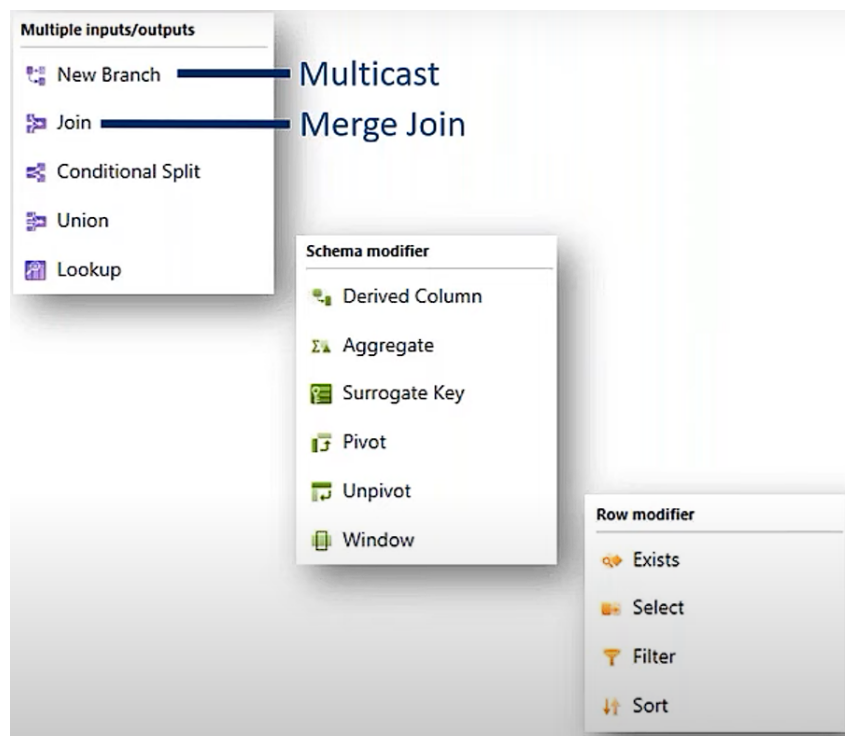
These transformations will **generate new data pipelines or merge into one** e.g. union of multiple data streams

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.

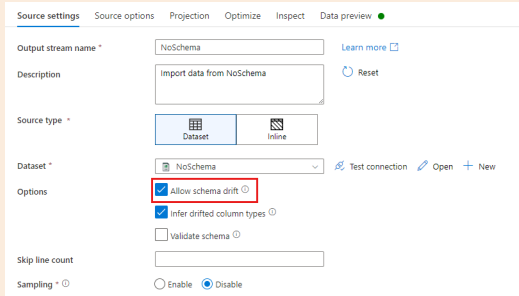
Here is a sample expression that can be used to create date directories and automatic partitioning:

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

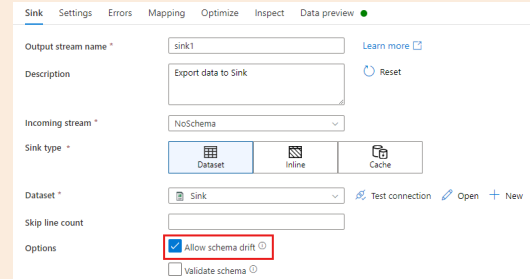


Schema Drift

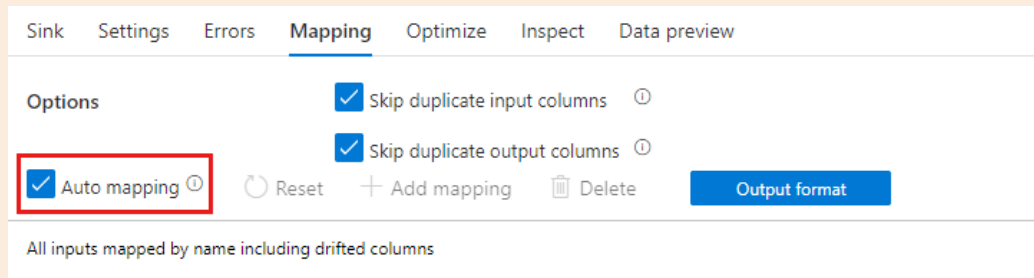
Schema drift is the case where your sources often change metadata. Fields, columns, and, types can be added, removed, or changed on the fly. Without handling schema drift, your data flow becomes vulnerable to upstream data source changes.



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.

Derived column's settings Optimize Inspect Data preview

Output stream name * MapDrifted [Learn more](#)

Description Creating/updating the columns 'movielid', 'title', 'genres' [Reset](#)

Incoming stream * NoSchema

Columns * ^⓪ [+ Add](#) [Clone](#) [Delete](#) [Open expression builder](#)

Column	Expression
movielid	toInteger(byName('movielid'))
title	toString(byName('title'))
genres	toString(byName('genres'))

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

Microsoft Azure | Data Factory | ctoadf1

Factory Resources powerquery1

Power Query functions

Queries [2]

UserQuery

Query settings

Applied steps

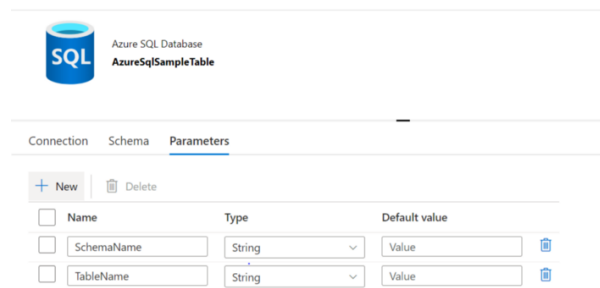
✓ Parameters

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.

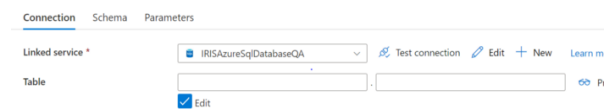
1) Dataset Parameters

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.

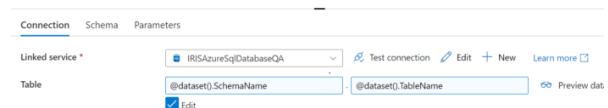
- You will need to create a dataset without mentioning the table name while creating it
- Parameters have to be added in the parameters tab. In the example for creating the Azure SQL dataset, we have added two parameters, one for the schema name and the other for the name of the table.



- 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.

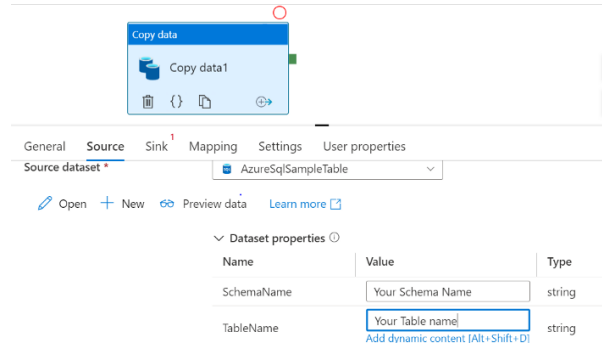


- We can click on the parameters we want to add by clicking on them. It will look like this



- After saving this dataset, You can pass table name and schema names

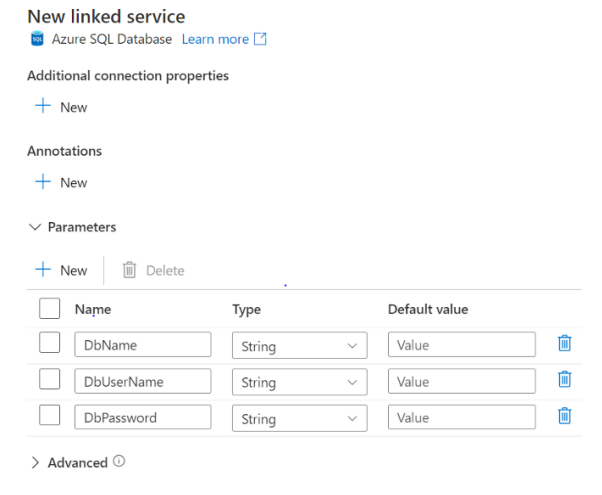
parameters inside the pipeline. ADF will ask for these values when we trigger/debug the pipeline.



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.

Edit linked service

[Azure SQL Database](#) [Learn more](#)

From Azure subscription
 Enter manually

Fully qualified domain name *

Add dynamic content [Alt+Shift+D]

Database name *

Authentication type *

SQL authentication

User name *

Password
 Azure Key Vault

Password *

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

Azure SQL Database
AzureSqlTable5

Connection Schema **Parameters**

+ New Delete

Name	Type	Default value
Schema	String	Value
Table	String	Value
dbName	String	Value
dbUserName	String	Value
dbPassword	String	.Value

- 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.

linked service *

AzureSqlSampleDatabase Test connection Edit + New Learn more

Linked service properties

Name	Value	Type
DbName	@{dataset().dbName}	string
DbUserName	@{dataset().dbUserName}	string
DbPassword	@{dataset().dbPassword}	string

Table

@{dataset().Schema} @{dataset().Table}

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

Save the current template for reuse Copy data1

General Source Sink **Mapping** Settings User properties

Source dataset *

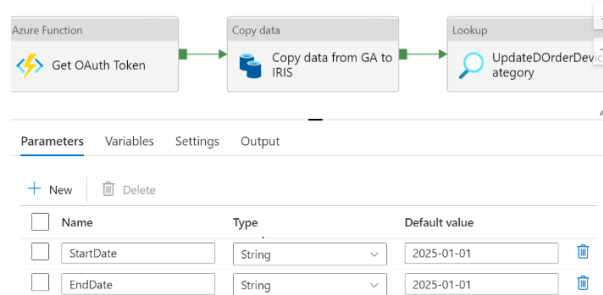
AzureSqlTable5 Open + New Preview data Learn more

Dataset properties

Name	Value	Type
Schema	Value	string
Table	Value	string
dbName	Value	string
dbUserName	Value	string
dbPassword	Value	string

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>`



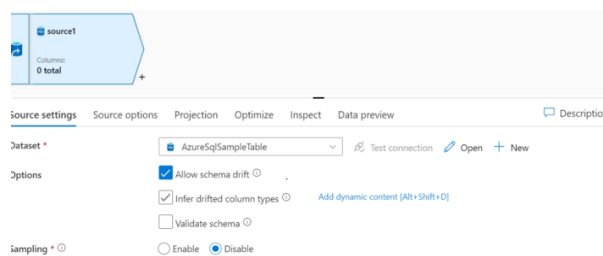
4) Parameters in Mapping Data Flow

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

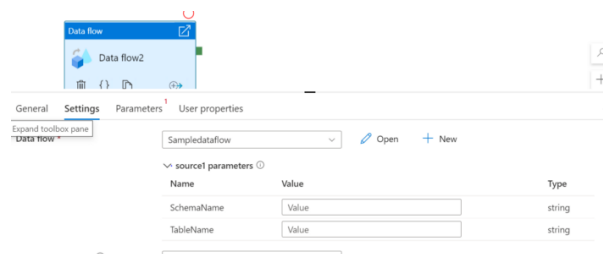
- Use the pipeline control flow expression language to set a dynamic value.
- Use the data flow expression language to set a dynamic value.
- Use either expression language to set a static literal value.

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

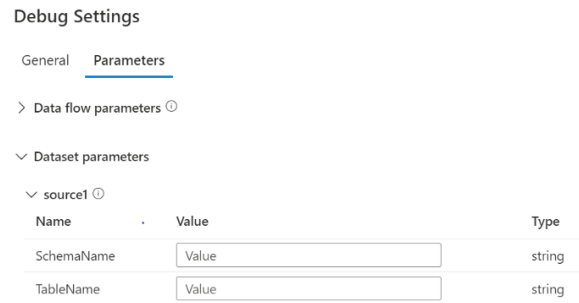
- Data flow is one of the activities in ADF pipeline, so the way to pass the parameters to it is the same as passing pipeline parameters above.
- When we create a dataflow we can select any parameterized dataset, for example, we have selected the dataset from the DATASET PARAMETERS section below.



- 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.



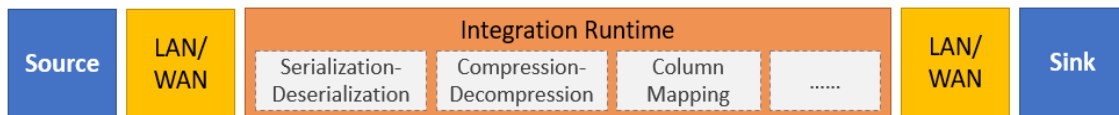
- If we want to access these during debugging the dataflow and not in the pipeline itself, we can see debug settings beside it. Inside the setting, we have the option to define the parameters.



[Link to an example showing Integration of a Notebook within Azure Synapse Pipelines](#)

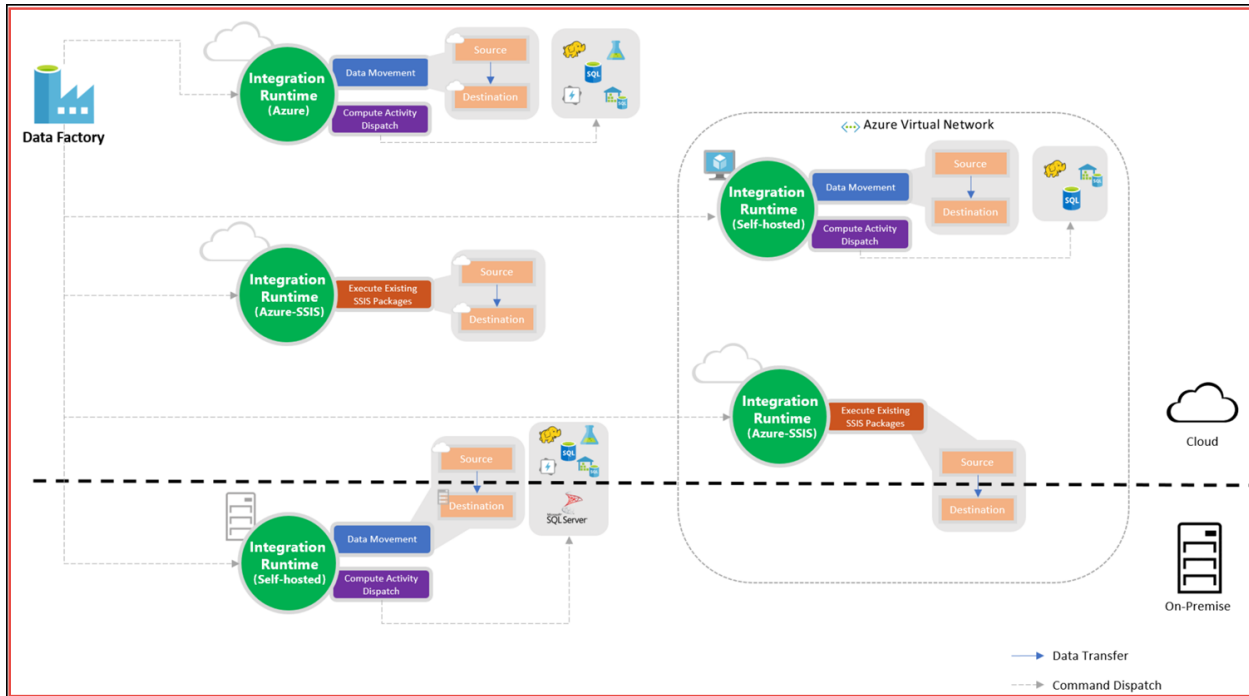
✓ Integration Runtime

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

Integration runtimes

The integration runtime (IR) is the compute infrastructure to provide the following [Learn more](#)

[+ New](#) [Refresh](#)

Showing 1 - 1 of 1 items

NAME ↑↓	TYPE ↑↓	SUB-TYPE ↑↓
AutoResolveIntegratio...	Azure	Public

[Create New IR](#)

Integration runtime setup

The Data Factory manages the integration runtime in Azure to connect to required data source/destination or external compute in public network. The compute resource is elastic allocated based on performance requirement of activities.

Name *
MySampleAzureIR

Description
Enter description here...

Type
Azure

Virtual network configuration (Preview)
 Disable Enable
 This data factory is not yet enabled with Virtual Network feature. [Request here to opt-in](#)

Region *
Auto Resolve

Data flow run time

Compute type *
General purpose

Core count *
4 (+ 4 Driver cores)

Time to live
0 minutes

Billing for data flows is based upon the type of compute you select and the number of cores selected per hour. If you set a TTL, then the minimum billing time will be that amount of time. Otherwise, the time billed will be based on the execution time of your data flows and the time of your debug sessions. Note that debug sessions will incur a minimum of 60 minutes of...

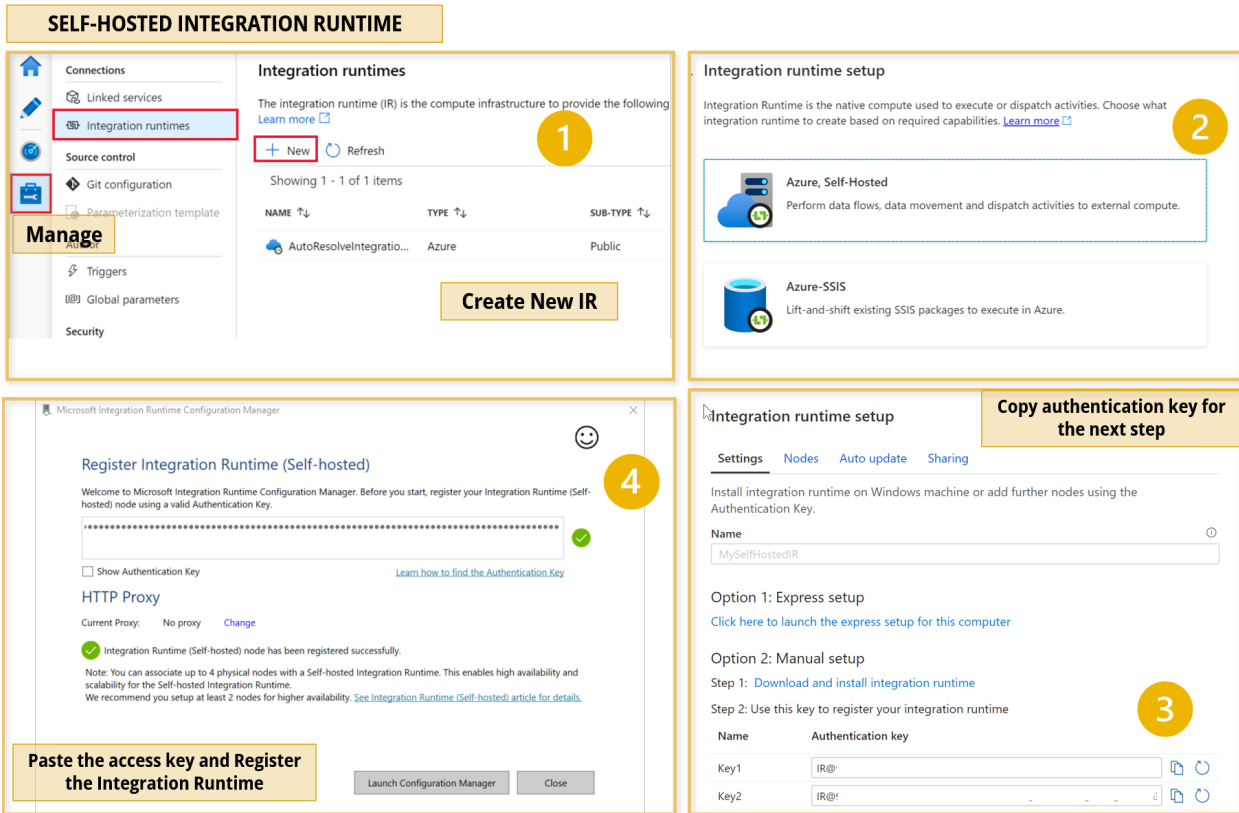
[Create](#) [Back](#) [Cancel](#)

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

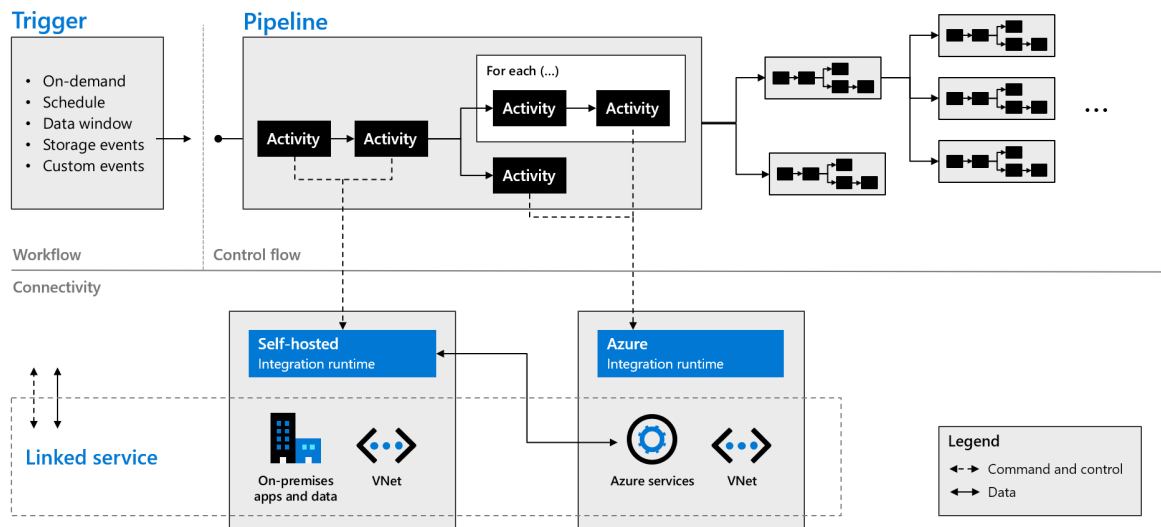
Supports SSIS package execution

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.

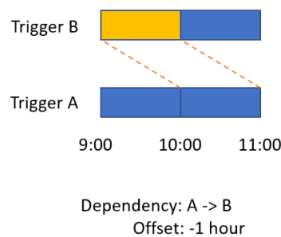
2) Tumbling Window Trigger

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.

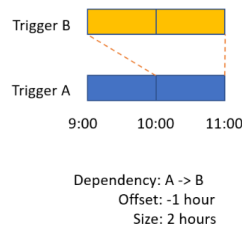
Consider the scenario in which you need to replicate data from a Database into a Data Lake on a regular basis, and you want to keep it in separate files or folders for every hour or day.

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.

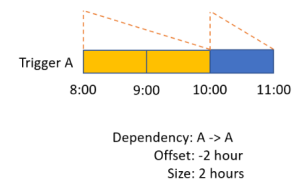
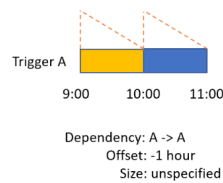
Dependency Offset

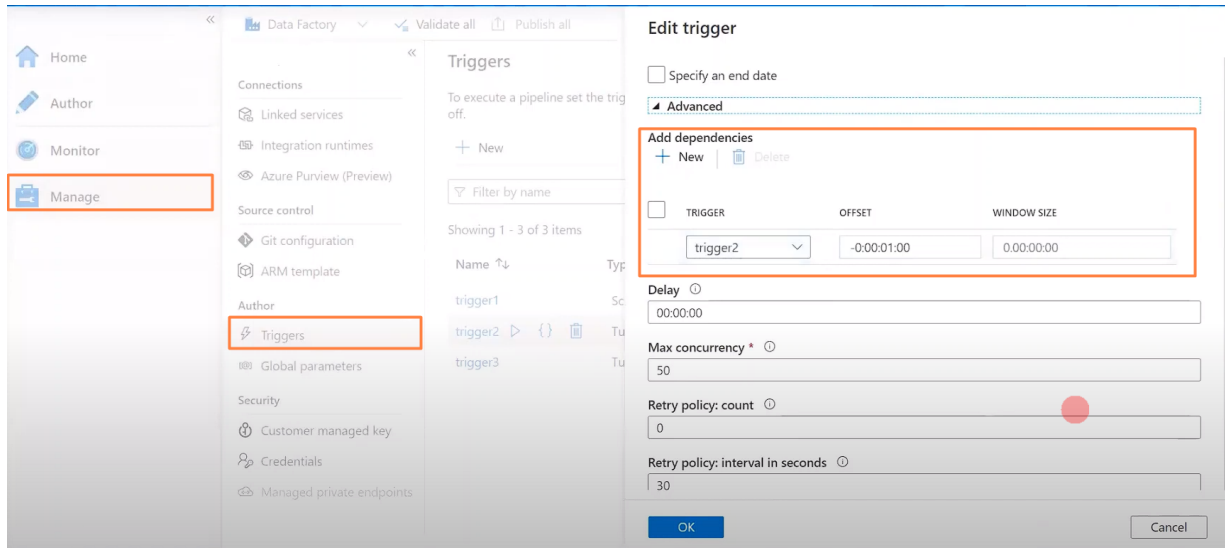


Dependency Size



Self Dependency





3) Event-based Trigger

The Event-based Azure Data Factory Trigger runs Data Pipelines **in 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

New trigger

Name *
trigger2

Description

Type *
Storage events

Account selection method *
 From Azure subscription Enter manually

Azure subscription *

Storage account name *

Container name *

Blob path begins with *

Blob path ends with *

Event *
 Blob created Blob deleted

Ignore empty blobs *
 Yes No

Annotations

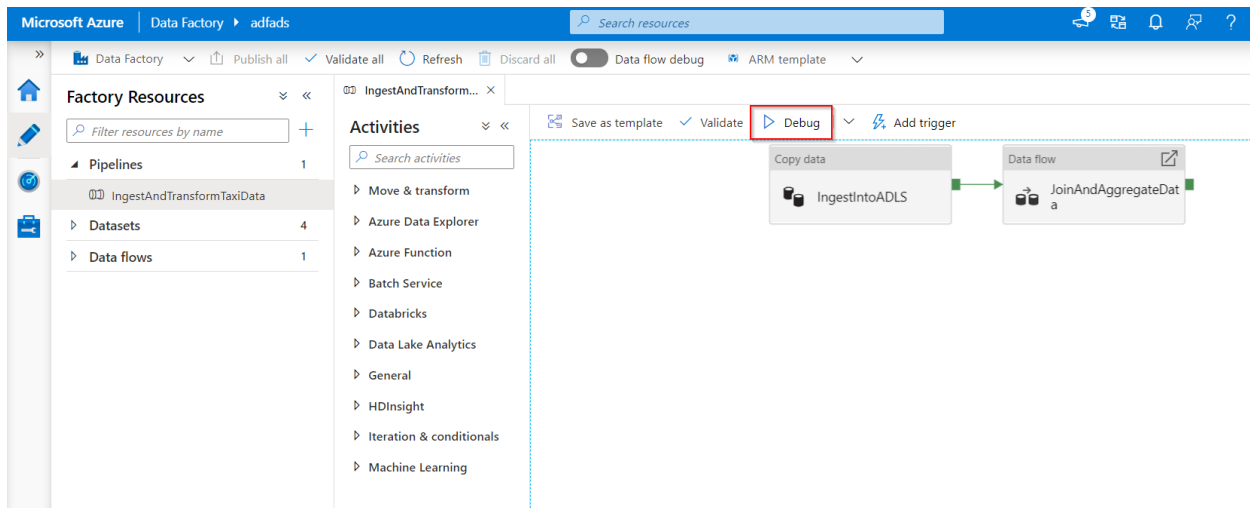
Activated *

✓ 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)

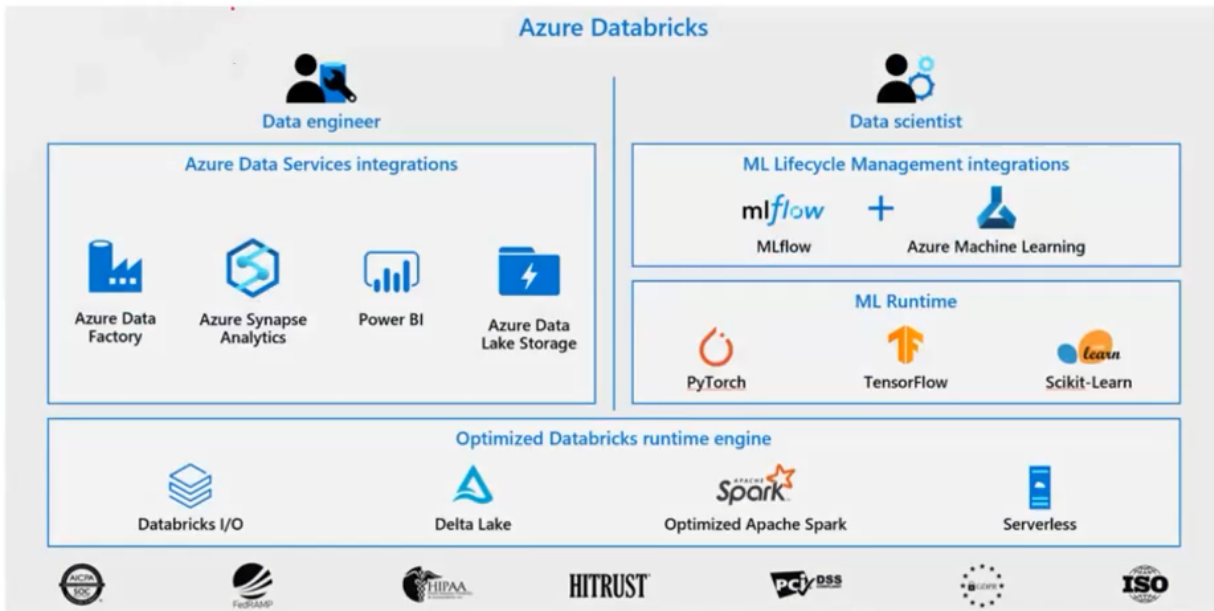
Azure Data Factory integrates with Azure DevOps and GitHub to allow easy source control and effective continuous integration and delivery. Azure Data Factory also offers a variety of both visual and programmatic monitoring services to also support the monitoring of your pipelines.

[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?

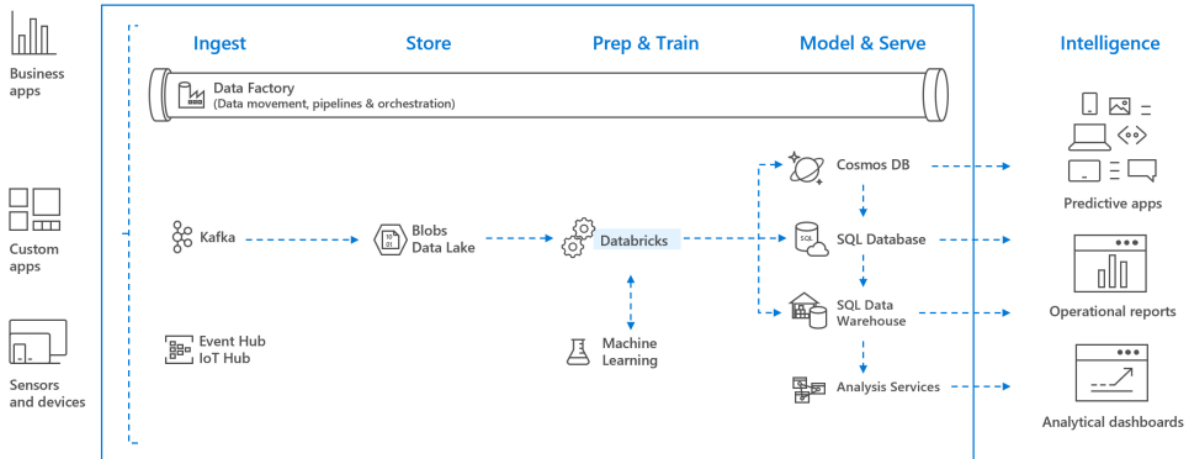
Azure Databricks is a wrapper around Apache Spark that simplifies the provisioning and configuration of a Spark cluster in a GUI interface

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.



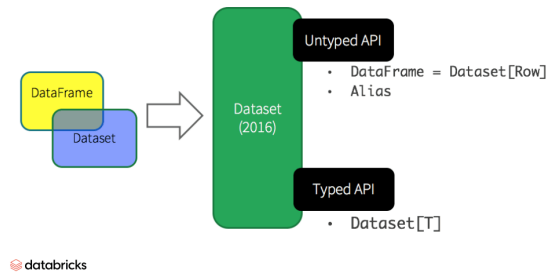
Azure Databricks Architecture

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.

Unified Apache Spark 2.0 API



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.

Create Cluster Free trial ends in 14 days. [Upgrade](#)

New Cluster Cancel Create Cluster **2-8 Workers:**28-112 GB Memory, 8-32 Cores, 1.5-6 DBU
1 Driver:14 GB Memory, 4 Cores, 0.75 DBU [?](#)

Cluster Name Please enter a cluster name

Cluster Mode [?](#)
 | v

Databricks Runtime Version [?](#) [Learn more](#)
 | v

Note Databricks Runtime 8.x uses Delta Lake as the default table format. [Learn more](#)

Autopilot Options

Enable autoscaling [?](#)

Terminate after minutes of inactivity [?](#)

Worker Type [?](#) 14 GB Memory, 4 Cores, 0.75 DBU | v

Min Workers Max Workers

⚠ Spot instances [?](#)

New Configure separate pools for workers and drivers for flexibility. [Learn more](#)

Driver Type 14 GB Memory, 4 Cores, 0.75 DBU | v

▶ Advanced Options

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.

Jobs / Create Fresh new look Free trial ends in 14 days. [Upgrad](#)

Cancel Create

Job must have a name

Runs Configuration

task

Type * /Shared/test 🔗 | 📁

Notebook | ▾

Cluster * ?

New Job Cluster (126.00 GB | 36 Cores | DBR 8.3 | Spark 3.1.1 | Scal... Edit | ▾

New Job Cluster Edit

126.00 GB | 36 Cores | DBR 8.3 | Spark 3.1.1 | Scala 2.12

Existing All-Purpose Clusters

● appcluster 🔗

14.00 GB | 4 Cores | DBR 8.3 | Spark 3.1.1 | Scala 2.12

Max

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

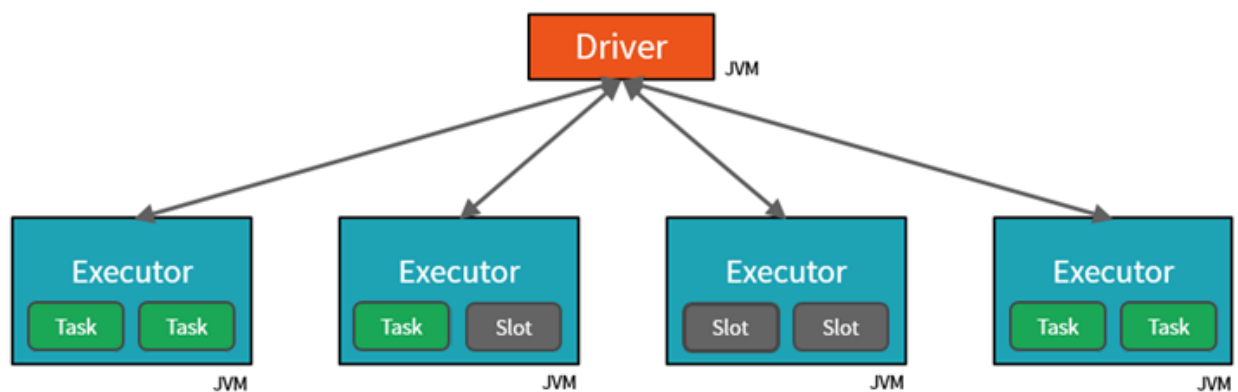
In Databricks, the notebook interface is typically the driver program. **SparkContext**, an object of the driver program runs the main function, creates distributed datasets on the cluster, applies parallel operations to the cluster nodes, and then collects the results of the operations.

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.

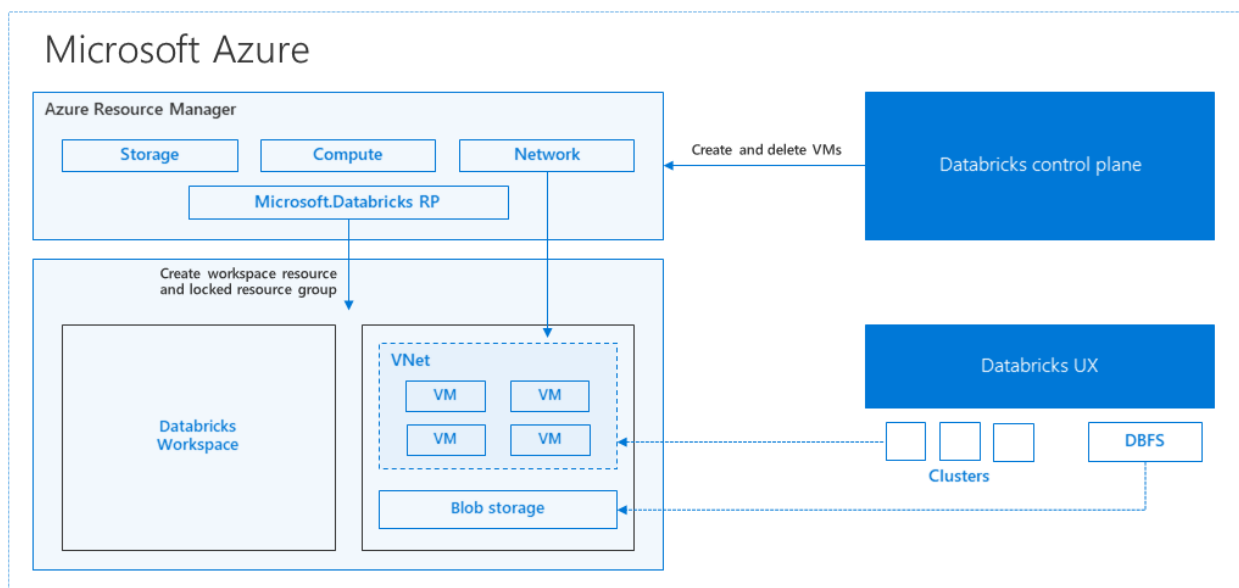
✓ How Azure manages cluster resources

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:

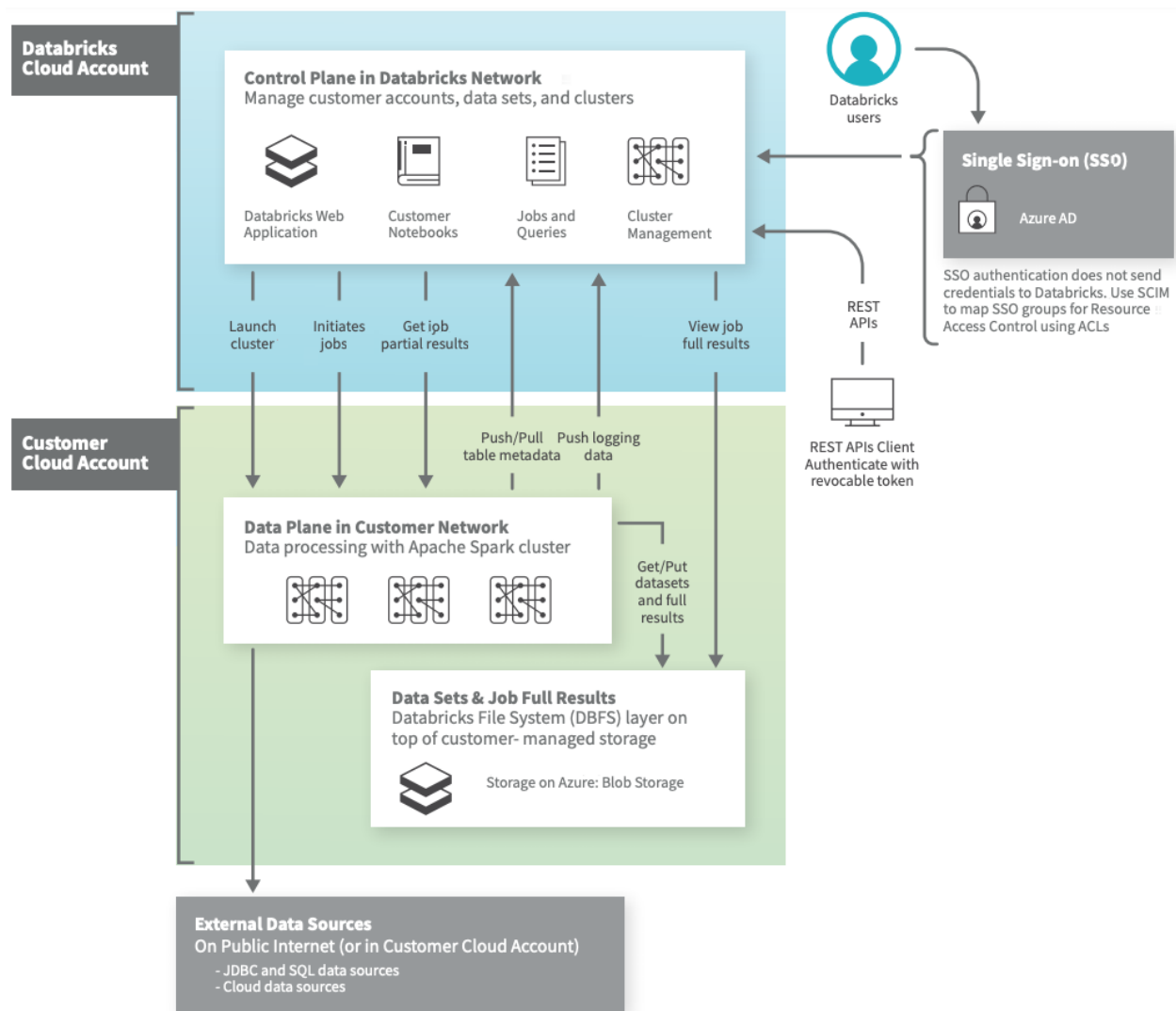
- The first level of parallelization is the **executor** - a Java virtual machine (JVM) running on a worker node, typically, one instance per node.
- The second level of parallelization is the **slot** - the number of which is determined by the number of cores and CPUs of each node.
- Each executor has 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
dutils.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["C
ategory"]=="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', format='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 rows
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 object in the Delta Lake API, which supports update, delete, and merge operations. For example, you could use the following code to update the price column for all rows with a category column value of "Accessories"

from delta.tables import *
from pyspark.sql.functions import *

# Create a deltaTable object
deltaTable = DeltaTable.forPath(spark, delta_table_path)

# Update the table (reduce 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_path)

# OR

df = spark.read.format("delta").option("timestampAsOf", '2022-01-01').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 table
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 SQL
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) \
    .tableName("default.ManagedProducts") \
    .addColumn("Productid", "INT") \
    .addColumn("ProductName", "STRING") \
    .addColumn("Category", "STRING") \
```

```
.addColumn("Price", "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.

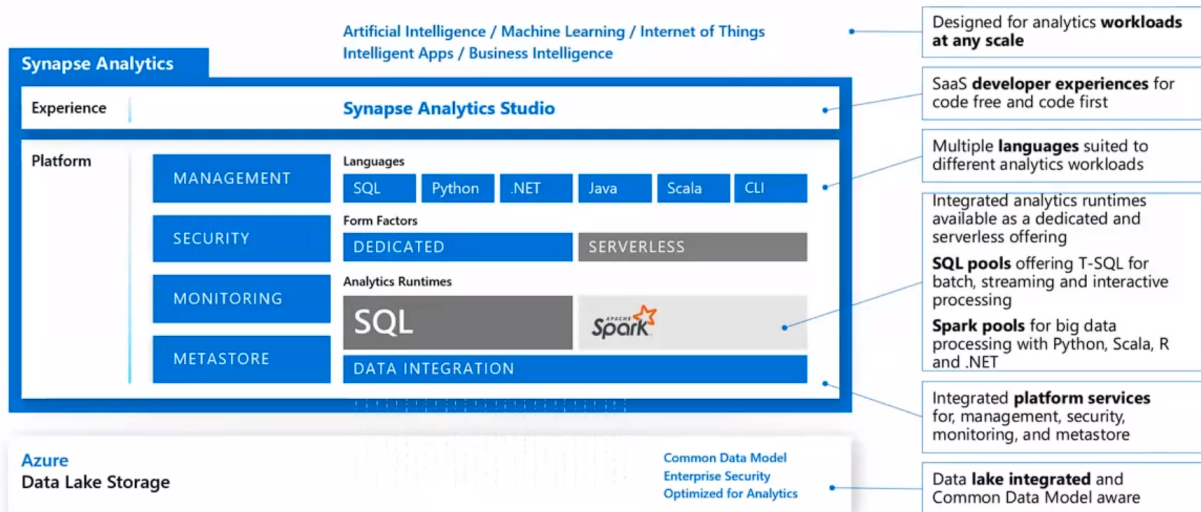
✓ Azure Synapse Analytics - OLAP

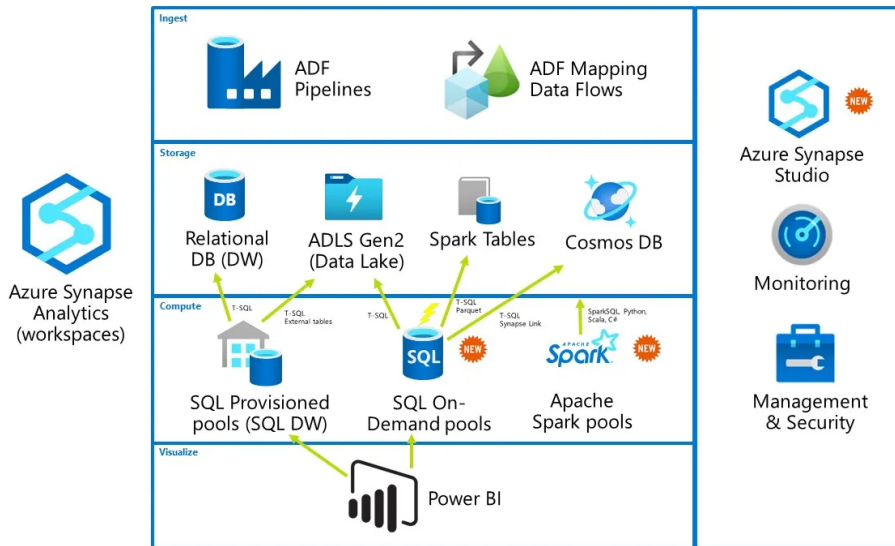
Run analytics at a massive scale by using a cloud-based enterprise data warehouse that takes advantage of massively parallel processing (MPP) to run complex queries quickly across petabytes of data.

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**.

Azure Synapse Analytics

Limitless analytics service with unmatched time to insight





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.

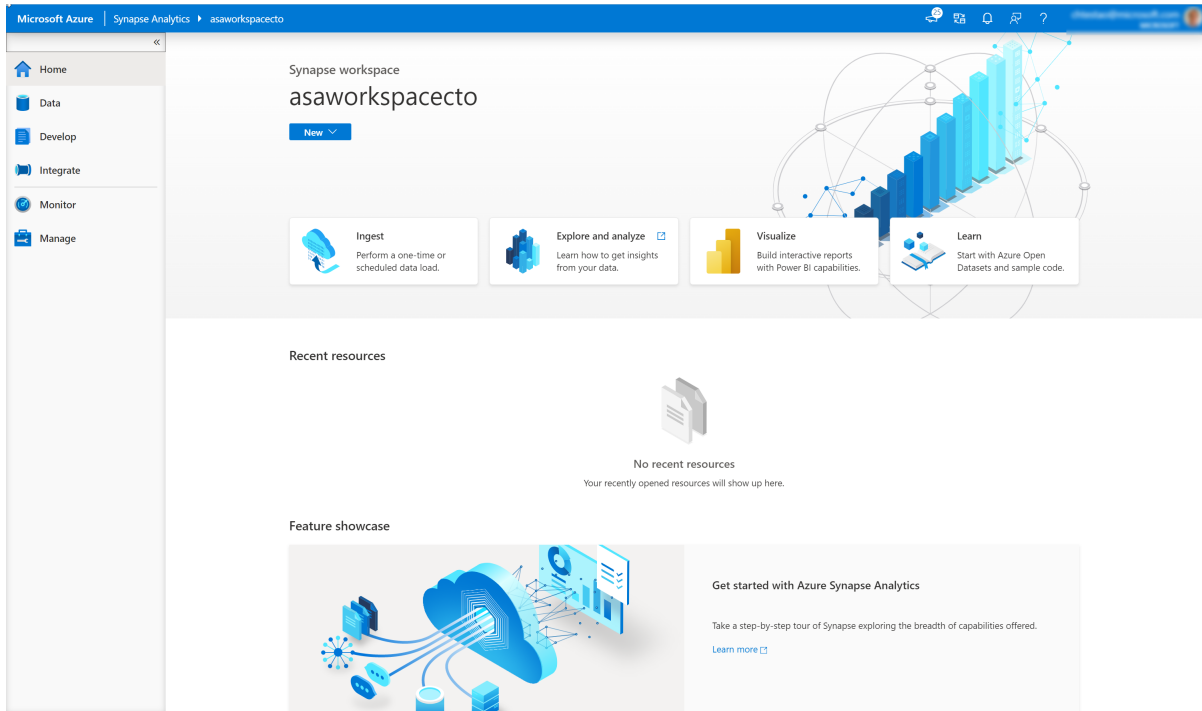
6) Synapse Studio

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 Studio



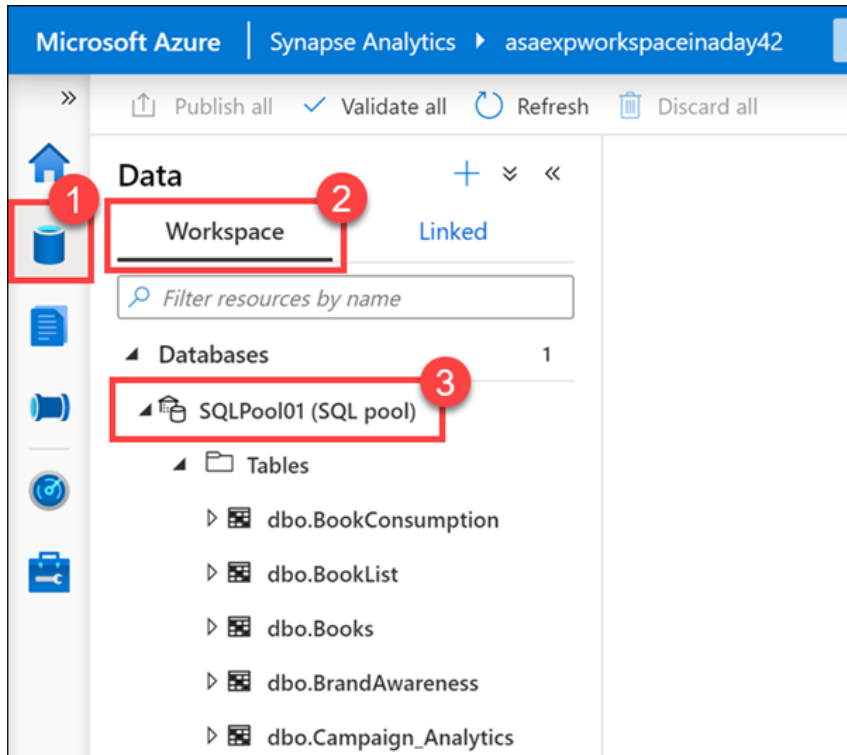
1) Data Hub

The Data hub is where you access your provisioned SQL pool databases and SQL serverless databases in your workspace, as well as external data sources, such as storage accounts and other linked services.

Under the **Workspace (2)** tab of the **Data hub (1)**, expand the **SQLPool01 (3)** SQL pool underneath **Databases**.

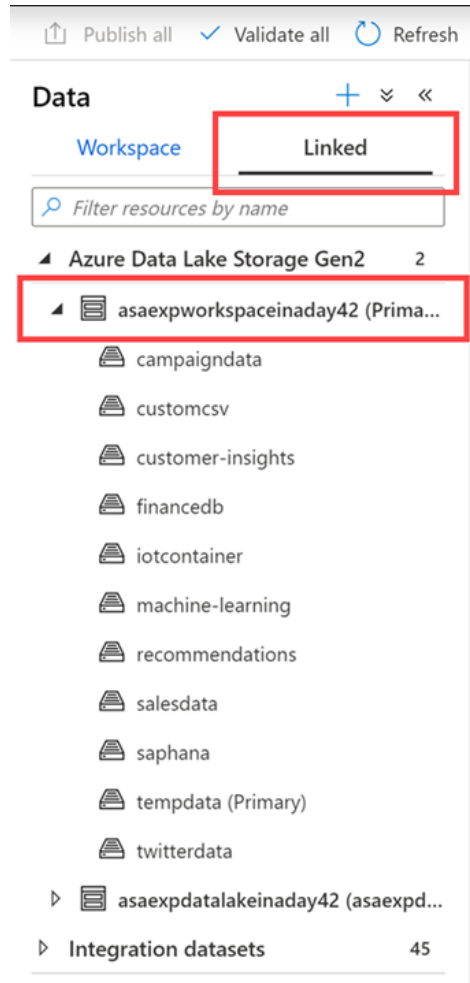
Expand **Tables** and **Programmability/Stored procedures**.

The **tables** listed under the SQL pool store data from multiple sources, such as SAP Hana, Twitter, Azure SQL Database, and external files copied over from an orchestration pipeline. Synapse Analytics gives us the ability to combine these data sources for analytics and reporting, all in one location.



You will also see familiar database components, such as **stored procedures**. You can execute the stored procedures using T-SQL scripts, or execute them as part of an orchestration pipeline.

Select the **Linked** tab, expand the **Azure Data Lake Storage Gen2** group, then expand the **primary storage** for the workspace.



Every Synapse workspace has a primary ADLS Gen2 account associated with it. This serves as the **data lake**, which is a great place to store flat files, such as files copied over from on-premises data stores, exported data or data copied directly from external services and applications, telemetry data, etc. Everything is in one place.

In our example, we have several containers that hold files and folders that we can explore and use from within our workspace. Here you can see marketing campaign data, CSV files, finance information imported from an external database, machine learning assets, IoT device telemetry, SAP Hana data, and tweets, just to name a few.

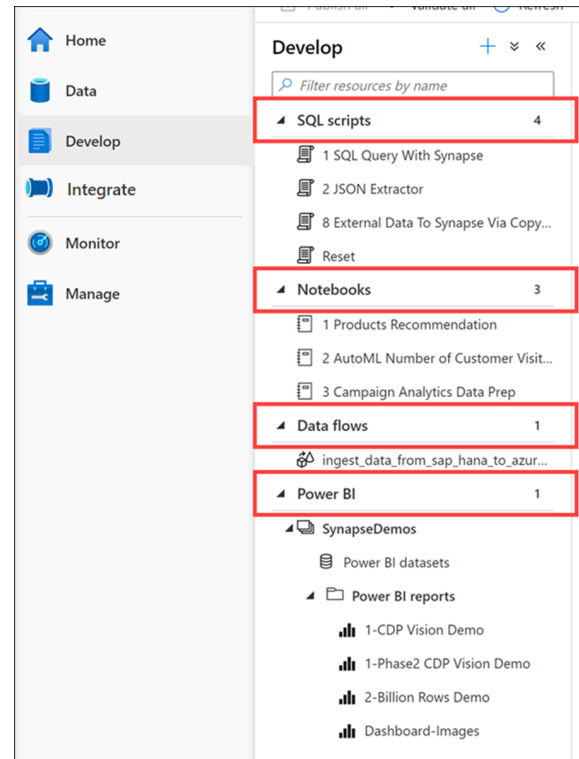
2) Develop

Expand each of the groups under the Develop menu. The Develop hub in our sample environment contains examples of the following artifacts:

- **SQL scripts** contains T-SQL scripts

that you publish to your workspace. Within the scripts, you can execute commands against any of the provisioned SQL pools or on-demand SQL serverless pools to which you have access.

- **Notebooks** contain Synapse Spark notebooks used for data engineering and data science tasks. When you execute a notebook, you select a Spark pool as its compute target.
- **Data flows** are powerful data transformation workflows that use the power of Apache Spark but are authored using a code-free GUI.
- **Power BI** reports can be embedded here, giving you access to the advanced visualizations they provide without ever leaving the Synapse workspace.



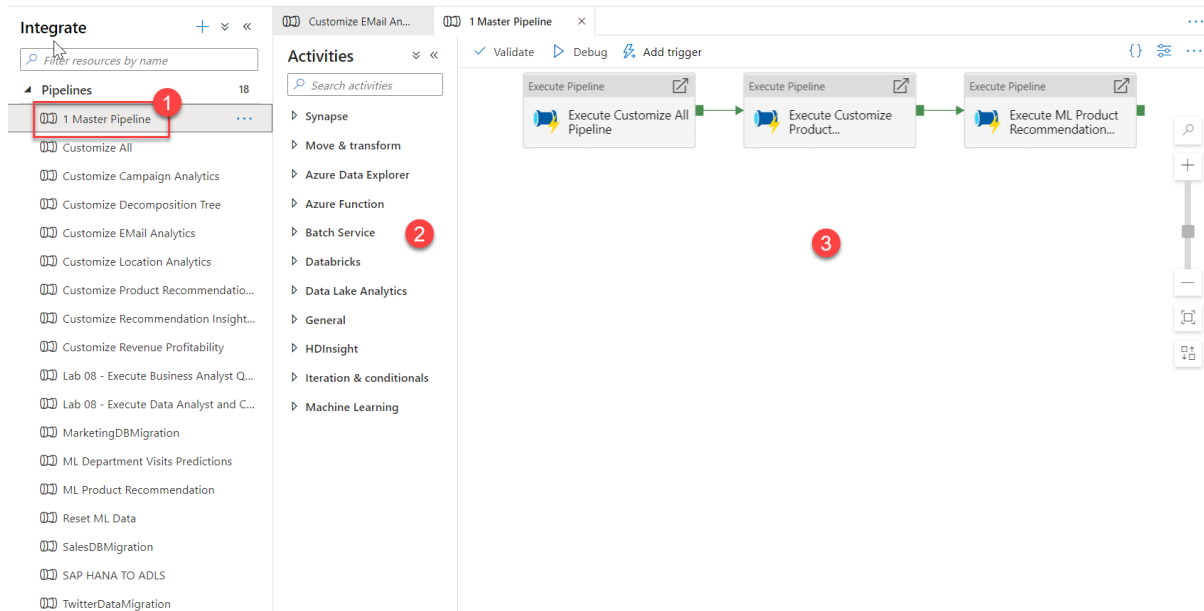
3) Integrate

Manage integration pipelines within the Integrate hub. If you are familiar with Azure Data Factory, then you will feel at home in this hub. The pipeline creation experience is the same as in ADF, which gives you another powerful integration built into Synapse Analytics, removing the need to use Azure Data Factory for data movement and transformation pipelines.

Expand Pipelines and select **Master Pipeline (1)**. Point out the **Activities (2)** that can be added to the pipeline, and show the **pipeline canvas (3)** on the right.

This Synapse workspace contains 16 pipelines that enable us to orchestrate data movement and transformation steps over data from several sources.

The **Activities** list contains many activities that you can drag and drop onto the pipeline canvas on the right.

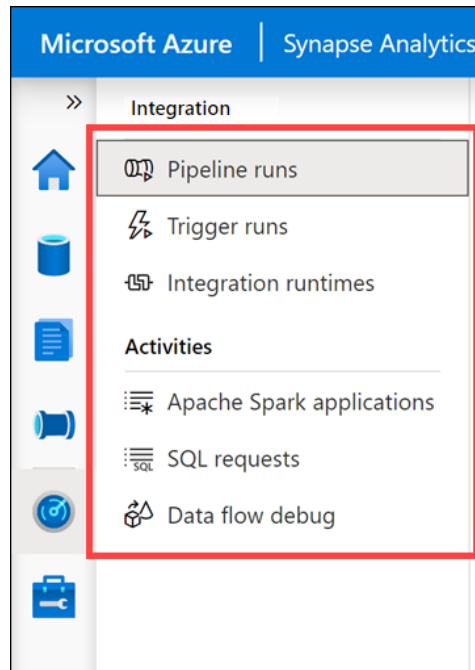


4) Monitor

The Monitor hub is your first stop for debugging issues and gaining insight on resource usage. You can see a history of all the activities taking place in the workspace and which ones are active now.

Show each of the monitoring categories grouped under Integration and Activities.

- **Pipeline runs** shows all pipeline run activities. You can view the run details, including inputs and outputs for the activities, and any error messages that occurred. You can also come here to stop a pipeline, if needed.
- **Trigger runs** shows you all pipeline runs caused by automated triggers. You can create triggers that run on a recurring schedule or tumbling window. You can also create event-based triggers that execute a pipeline any time a blob is created or deleted in a storage container.



- **Integration runtimes** shows the status of all self-hosted and Azure integration runtimes.
- **Apache Spark applications** shows all the Spark applications that are running or have run in your workspace.
- **SQL requests** shows all SQL scripts executed either directly by you or another user, or executed in other ways, like from a pipeline run.
- **Data flow debug** shows active and previous debug sessions. When you author a data flow, you can enable the debugger and execute the data flow without needing to add it to a pipeline and trigger an execute. Using the debugger speeds up and simplifies the development process. Since the debugger requires an active Spark cluster, it can take a few minutes after you enable the debugger before you can use it.

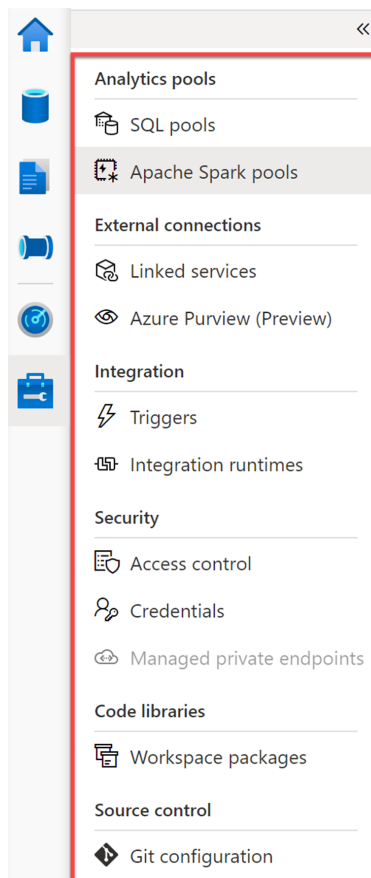
5) Manage

Show each of the management categories grouped under Analytics pools, External connections, Integration, and Security.

- **SQL pools.** Lists the provisioned SQL pools and on-demand SQL serverless pools for the workspace. You can add new pools or hover over a SQL pool

to **pause** or **scale** it. You should pause a SQL pool when it's not being used to save costs.

- **Apache Spark pools.** Lets you manage your Spark pools by configuring the auto-pause and auto-scale settings. You can provision a new Apache Spark pool from this blade.
- **Linked services.** Enables you to manage connections to external resources. Here you can see linked services for our data lake storage account, Azure Key Vault, Power BI, and Synapse Analytics. **Task:** Select **+ New** to show how many types of linked services you can add.



- **Azure Purview (Preview).** Provides integration with Azure Purview to provide data governance and lineage within Azure Synapse Analytics.
- **Triggers.** Provides you a central location to create or remove pipeline triggers. Alternatively, you can add triggers from the pipeline.

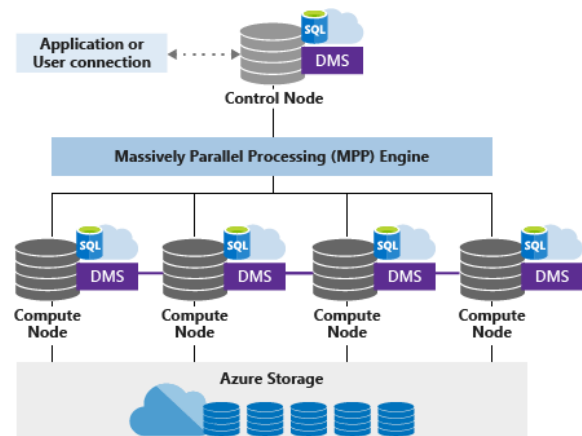
- **Integration runtimes.** Lists the IR for the workspace, which serves as the compute infrastructure for data integration capabilities, like those provided by pipelines. **Task:** Hover over the integration runtimes to show the monitoring, code, and delete (if applicable) links. Click on a **code link** to show how you can modify the parameters in JSON format, including the TTL (time to live) setting for the IR.
- **Access control.** This is where you go to add and remove users to one of three security groups: workspace admin, SQL admin, and Apache Spark for Azure Synapse Analytics admin.
- **Credentials.** Contains objects that hold authentication information that can be used by Azure Synapse Analytics.
- **Managed private endpoints.** This is where you manage private endpoints, which use a private IP address from within a virtual network to connect to an Azure service or your own private link service. Connections using private endpoints listed here provide access to Synapse workspace endpoints (SQL, SqlOndemand and Dev).
- **Workspace packages.** Workspace packages can be custom code or a specific version of an open-source library that you would like to use in your Apache Spark pools held in the Azure Synapse Analytics Workspace.
- **Git configuration.** Enables you to connect your workspace to a Git repository to enable source control

▼ ✓ SYNAPSE SQL (Important)

✓ Azure Synapse Architecture (dedicated SQL Pool)

When a user raises a work/query, the following happens:

Step 1: Applications connect and issue T-SQL commands to a **Control node**. The Control node is the brain of the architecture. It is the front end that interacts with all applications and connections. This node hosts the distributed query engine (MPP), which



optimizes the query for parallel processing.

Step 2: Control node provides commands to multiple **compute nodes** (the number depends on the option we selected during setup - DWU) which will work in parallel to compute the query. The Compute nodes provide the **computational power**. Distributions map to Compute nodes for processing.

Step 3: The **Data Movement Service (DMS)** is a system-level internal service that moves data across the nodes as necessary to run queries in parallel and return accurate results. The number of compute nodes ranges from 1 to 60 and is determined by the service level for Synapse SQL.

Step 4: A key architectural component of dedicated SQL pools is the decoupled storage that is segmented into 60 parts. The data is shared by these **distributions** in the data layer to optimize the work performance. ***Distribution is the basic unit of storage and parallel queries process these distributed data.***

When Synapse SQL runs a query, the work is divided into 60 smaller queries that run in parallel. Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. Since there are 60 storage segments and a maximum of 60 MPP compute nodes within the highest performance configuration of SQL Pools, a 1:1 file to compute node to storage segment may be viable for ultra-high workloads.

With decoupled storage and compute, when using a dedicated SQL pool, we can scale each of these independently.

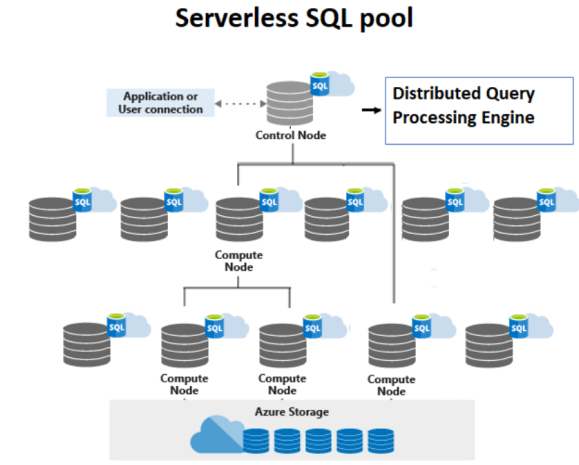


**Azure Storage is divided into 60 segments called distributions
Additionally, we can have max 60 compute nodes for computation
So at the highest service level, each compute node will get 1
distribution to work on.
In other cases, each compute node can have more than 1 distributions
to process.**

For best practices, check this [link](#).

Azure Synapse Architecture (serverless SQL Pool)

For a serverless SQL pool, being serverless, scaling is done automatically to accommodate query resource requirements. As topology changes over time by adding, removing nodes, or failovers, it adapts to changes and makes sure your query has enough resources and finishes successfully. For example, the following image shows a serverless SQL pool using four compute nodes to execute a query.



If you use Apache Spark for Azure Synapse in your data pipeline, for data preparation, cleansing, or enrichment, you can query external Spark tables you've created in the process, directly from the serverless SQL pool. (more on this later)

Serverless SQL Pool	Dedicated SQL Pool
Perform unplanned or ad-hoc analysis work	Build data warehouse
Only create external data tables	If one needs to persist the data
Charged based on the amount of data processes (as there's no underlying infrastructure)	Charges based on DWU (Data Warehouse Units)

For more detailed differences check this [link](#). Also for best practices, check [this](#).

✓ Designing a data warehouse ([link](#))

Transactional Processing (OLTP) FACT & DIMENSION TABLE	Analytical Processing (OLAP) FACT & DIMENSION TABLE
Used for storing individual entries and analysis on small sets of data	Analyses large batches of data
Access to recent data (maybe only 2022 data)	Access to older data going back years (all historical data in order to perform analysis)
Updates data (individual transactions are inserted, delete, and update)	Optimized for reading operations (only bulk data should be uploaded, not optimized for individual entries)

Transactional Processing (OLTP) FACT & DIMENSION TABLE	Analytical Processing (OLAP) FACT & DIMENSION TABLE
Normalization concept applies and architecture is generally SNOWFLAKE schema	Strict adherence to normalization is not followed, STAR schema is followed (i.e. one or more dimension tables from SQL Database can be merged and/or appended to get a single dimension table in SQL Data Warehouse)
Faster real-time access	Long-running jobs
Usually a single data source	Multiple data sources Dimension tables can be connected from SQL Database, CSV files, and more for analysis purposes



Ideally, try to **replace NULL values** with some default values **in the dimension tables**, as not doing this can give undesired results while using reporting tools.

A **Fact table can have NULL values** with the exception of the key columns which will be used for joining to the dimension table

Data integrity constraints

Dedicated SQL pools in Synapse Analytics don't support *foreign key* and *unique* constraints as found in other relational database systems like SQL Server. This means that jobs used to load data must maintain uniqueness and referential integrity for keys, without relying on the table definitions in the database to do so.

▼ Transfer Data to a Dedicated SQL pool

In order to proceed ahead, we will need to set up a **staging area** where data from the SQL database is first stored before it is moved into Azure Synapse.

```
-- Lab - Transfer data to our SQL Pool

-- First let's ensure we have the tables defined in the SQL pool

CREATE TABLE [dbo].[SalesFact](
    [ProductID] [int] NOT NULL,
    [SalesOrderID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
```

```

[OrderQty] [smallint] NOT NULL,
[UnitPrice] [money] NOT NULL,
[OrderDate] [datetime] NULL,
[TaxAmt] [money] NULL
)

CREATE TABLE [dbo].[DimCustomer](
[CustomerID] [int] NOT NULL,
[StoreID] [int] NOT NULL,
[BusinessEntityID] [int] NOT NULL,
[StoreName] varchar(50) NOT NULL
)

CREATE TABLE [dbo].[DimProduct](
[ProductID] [int] NOT NULL,
[ProductModelID] [int] NOT NULL,
[ProductSubcategoryID] [int] NOT NULL,
[ProductName] varchar(50) NOT NULL,
[SafetyStockLevel] [smallint] NOT NULL,
[ProductModelName] varchar(50) NULL,
[ProductSubCategoryName] varchar(50) NULL
)

SELECT * FROM [dbo].[SalesFact]
SELECT COUNT(*) FROM [dbo].[SalesFact]

SELECT * FROM [dbo].[DimCustomer]
SELECT COUNT(*) FROM [dbo].[DimCustomer]

SELECT * FROM [dbo].[DimProduct]
SELECT COUNT(*) FROM [dbo].[DimProduct]

-- If we need to drop the tables

DROP TABLE [dbo].[SalesFact]

DROP TABLE [dbo].[DimCustomer]

DROP TABLE [dbo].[DimProduct]

```

Go to Synapse Studio → Integrate → Copy Data tool → Run Once now → Create connection (define **source** settings) → Select Azure SQL Database → Fill details → Select the required tables for copying → Create connection (define **target** settings) → Select Azure Synapse → Fill details → Option to select column mapping (if we want to drop certain columns) → Set the staging account details in the **Settings** option → In the advanced option we have option to select the copying procedure (PolyBase, Copy Command or Bulk insert) → review and deploy the pipeline

Reading JSON Files

```
-- Lab - Reading JSON files

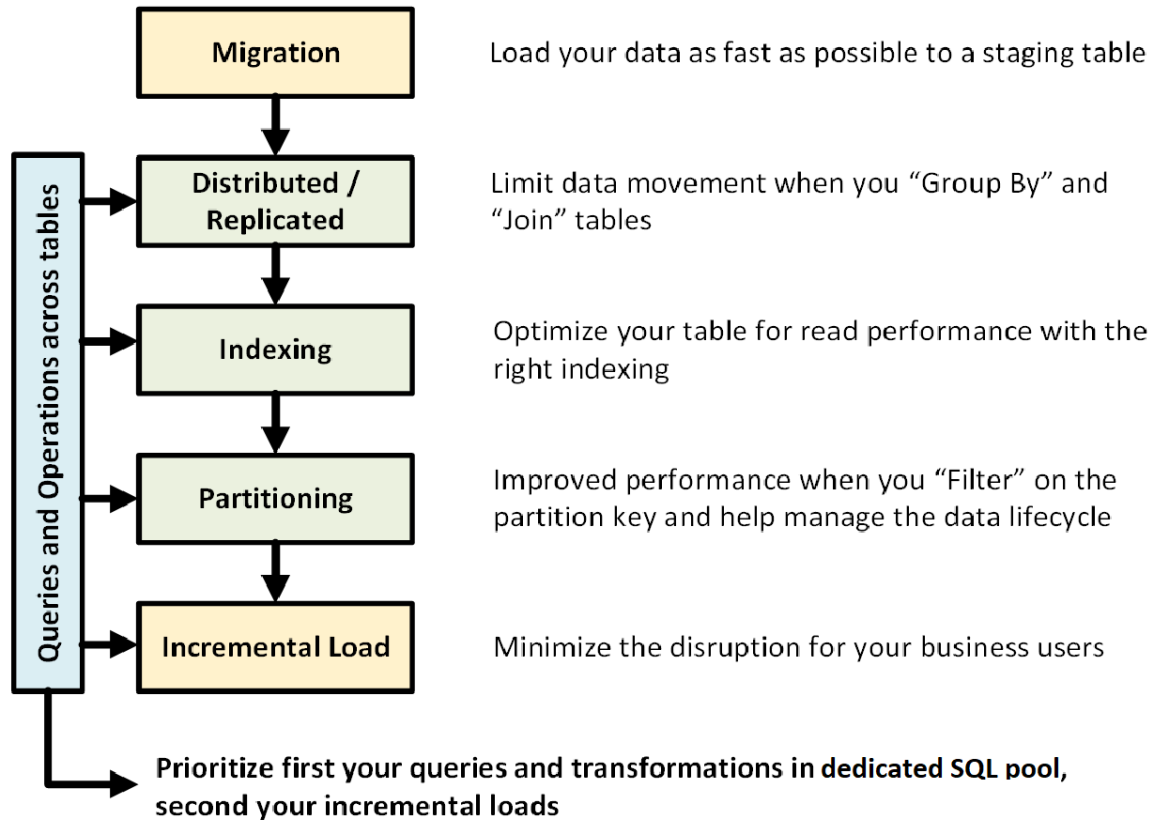
-- Here we are using the OPENROWSET Function

SELECT TOP 100
    jsonContent
FROM
    OPENROWSET(
        BULK 'https://appdatalake7000.dfs.core.windows.net/data/log.json',
        FORMAT = 'CSV',
        FIELDQUOTE = '0x0b',
        FIELDTERMINATOR = '0x0b',
        ROWTERMINATOR = '0x0a' --Ensure this is different from field terminator
    )
    WITH (
        jsonContent varchar(MAX)
    ) AS [rows]

-- The above statement only returns all as a single string line by line
-- Next we can cast to seperate columns

SELECT
    CAST(JSON_VALUE(jsonContent,'$.Id') AS INT) AS Id,
    JSON_VALUE(jsonContent,'$.Correlationid') As Correlationid,
    JSON_VALUE(jsonContent,'$.Operationname') AS Operationname,
    JSON_VALUE(jsonContent,'$.Status') AS Status,
    JSON_VALUE(jsonContent,'$.Eventcategory') AS Eventcategory,
    JSON_VALUE(jsonContent,'$.Level') AS Level,
    CAST(JSON_VALUE(jsonContent,'$.Time') AS datetimeoffset) AS Time,
    JSON_VALUE(jsonContent,'$.Subscription') AS Subscription,
    JSON_VALUE(jsonContent,'$.Eventinitiatedby') AS Eventinitiatedby,
    JSON_VALUE(jsonContent,'$.Resourcetype') AS Resourcetype,
    JSON_VALUE(jsonContent,'$.Resourcegroup') AS Resourcegroup
FROM
    OPENROWSET(
        BULK 'https://appdatalake7000.dfs.core.windows.net/data/log.json',
        FORMAT = 'CSV',
        FIELDQUOTE = '0x0b',
        FIELDTERMINATOR = '0x0b',
        ROWTERMINATOR = '0x0a'
    )
    WITH (
        jsonContent varchar(MAX)
    ) AS [rows]
```


What you should aim for



Synapse SQL dedicated pools have three different types of tables indexing based on how the data is stored.

- 1) **Clustered columnstore**
- 2) **Clustered index**
- 3) **Heap**

Synapse dedicated pools support sharding for all these table types. They provide three different ways to shard the data, as follows:

- 1) **Hash**
- 2) **Round-robin**
- 3) **Replicated**

These methods through which a SQL dedicated pool distributes data among its tables are also called distribution techniques. Sharding and distribution

techniques are overlapping technologies that are always specified together in SQL CREATE TABLE statements.

✓ Table Types (Sharding Patterns for Dedicated SQL Pool i.e Horizontal Partitioning)

In a dedicated SQL pool, data is already distributed across its 60 distributions, so we need to be careful in deciding if we need to further partition the data. For example, if we plan to partition the data further by the months of a year, we are talking about 12 partitions x 60 distributions = 720 sub-divisions. Each of these divisions needs to have at least 1 million rows; in other words, the table (usually a fact table) will need to have more than 720 million rows. So, we will have to be careful to not over-partition the data when it comes to dedicated SQL pools.

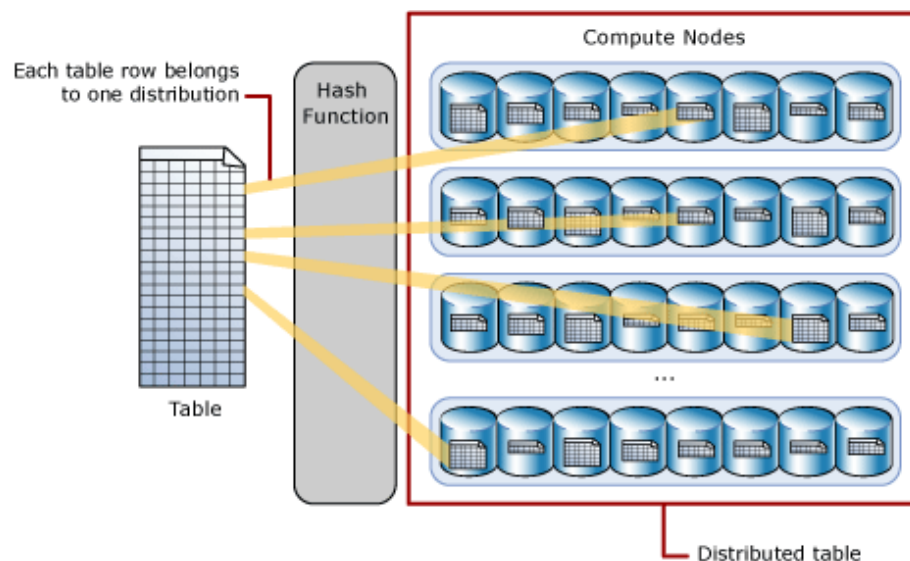
There are three different ways to distribute (shard) data among distributions :

1) Hash-distributed tables

(use this on a fact table with a hash column selected carefully)

Highest query performance for joins and aggregations on large tables

This works quicker if the query aggregation works on the hash column that we defined.



To shard data into a hash-distributed table, a **hash function** is used to **deterministically assign each row to one distribution**. In the table definition, one of

the columns is designated as the distribution column. The hash function uses the values in the distribution column to assign each row to a distribution deterministically. For eg, all the rows having category id less than 100 goes to one distribution, and so on.

When choosing the hash column, try to avoid columns having data skew as it would lead to uneven distribution of rows across the nodes. Also, avoid selecting the date column.

A quick way to check for data skew is to use `DBCC PDW_SHOWSPACEUSED`. The following SQL code returns the number of table rows that are stored in each of the 60 distributions. For balanced performance, the rows in your distributed table should be spread evenly across all the distributions.

```
- Find data skew for a distributed table
DBCC PDW_SHOWSPACEUSED('dbo.FactInternetSales');
```

Consider using a hash-distributed table when:

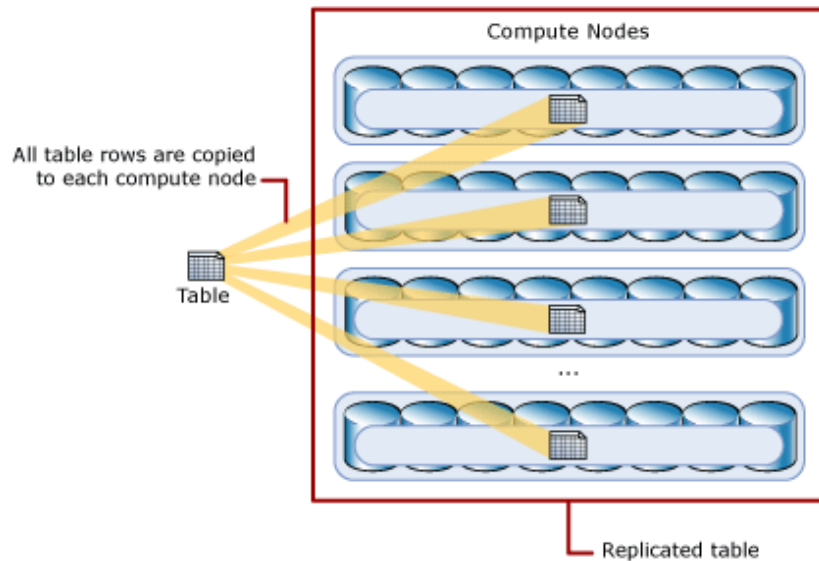
- The table size on the disk is more than 2 GB.
- The table has frequent insert, update, and delete operations.

2. Replicated tables

(use it for dimension tables that are smaller in size <2 GB)

Fastest query performance for small tables

Caches a full copy of the table on each compute node. Consequently, replicating a table removes the need to transfer data among compute nodes before a join or aggregation but incurs additional overhead.



Don't consider this table type if the table has a frequent insert, update, and delete operations as it will require a rebuild of the replicated table. A replicated table provides the fastest query performance for small tables which with compression should be less than 2GB as a starting point, static data can be larger.

3) Round-robin distributed tables

(default option during table creation)

Simplest table to create and delivers fast performance when used as a staging table for loads

A round-robin distributed table **distributes data evenly across the table** but without any further optimization. A distribution is first chosen at random and then buffers of rows are assigned to distributions sequentially. Joins on round-robin tables require **reshuffling** data, which **takes additional time** as it takes time to move data over from other nodes and collate all the rows together.

Consider this option if there are no joins performed on the tables or in the case when we don't have a clear candidate column for the hash distributed table.

Consider using the round-robin distribution for your table in the following scenarios:

- When getting started as a simple starting point since it is the default
- If there is no obvious joining key
- If there is no good candidate column for hash distributing the table
- If the table does not share a common join key with other tables

- If the join is less significant than other joins in the query
- When the table is a temporary staging table

Summary

Type of Distribution	Best Fit for...	Do not use when...
Replicated	<p>–Small dimension tables in a star schema with less than 2 GB of storage after the compression (Synapse does 5x compression). - Good for small lookup tables. -Good for dimension tables that are frequently joined with other big tables.</p>	<p>-Many write transactions are on the table (for example insert, delete and updates). -If you change the datawarehouse Units frequently. - You only use 2 -3 columns out of many columns in your tables. -you are indexing a replicated table.</p>
Round Robin (default)	<p>–Temporary /staging Table. –No obvious joining key candidate is found in the table or If your data doesn't frequently join with data from other tables. –When you cannot identify a single key to distribute your data. –Small dimension table.</p>	<p>–Performance is slow due to data movement</p>
Hash	<p>–Large Fact Tables or historical Transaction tables are good candidates. –Large dimension tables.</p>	<p>–The distribution key can not be updated –A nullable column is a bad candidate for any hash distributed table. –Fact tables that has a default value in a column is also not a good candidate to create a hash distributed table.</p>

To balance the parallel processing, select a distribution column or set of columns that:

- **Has many unique values.** The distribution column(s) can have duplicate values. All rows with the same value are assigned to the same distribution. Since there are 60 distributions, some distributions can have > 1 unique value while others may end with zero values.
- **Does not have NULLs, or has only a few NULLs.** For an extreme example, if all values in the distribution column(s) are NULL, all the rows are assigned to the

same distribution. As a result, query processing is skewed to one distribution, and does not benefit from parallel processing.

- **Is not a date column.** All data for the same date lands in the same distribution or will cluster records by date. If several users are all filtering on the same date (such as today's date), then only 1 of the 60 distributions do all the processing work.

✓ Indexing

In SQL-based systems, you might be required to access rows using values other than the primary key. In such cases, the query engine needs to scan all the rows to find the value we are looking for. Instead, if we can define a secondary index based on frequently searched column values, we could avoid the complete table scans and speed up the query. The secondary index tables are calculated separately from the primary indexes of the table, but this is done by the same SQL engine.

A well-designed indexing strategy can reduce disk I/O operations and consume less system resources therefore improving query performance, especially when using filtering, scans, and joins in a query.

1) Clustered Columnstore Index (no secondary index)

By default for an Azure Synapse Dedicated SQL pool table, a clustered columnstore index gets created automatically. This provides the **highest level of data compression and the best overall query performance**. In a normal SQL database, the data is stored row by row but in SQL DataWarehouse, it is stored column by column. Clustered columnstore tables will generally outperform a clustered index or heap tables and are usually the best choice for large tables.

However, this kind of index can't be created with columns that are of type varchar, nvarchar, varbinary. **Also clustered columnstore index is not ideal for small tables having less than 60 million rows and also for transient data.**



NOTE:

Since a columnstore index scans a table by scanning column segments of individual rowgroups, maximizing the number of rows in each rowgroup enhances query performance. A rowgroup can have a maximum of 1,048,576 rows. However, Columnstore indexes achieve good performance when rowgroups have at least 100,000 rows.

2) Clustered Index (allow secondary index, no compression)

Clustered index tables are row-based storage tables. They are usually faster for queries that need row lookups with highly selective filters on the clustered index column.

Clustered indexes may outperform clustered columnstore tables when a single row needs to be quickly retrieved. For queries where a single or very few row lookup is required to perform with extreme speed, consider a clustered index or nonclustered secondary index.

The disadvantage to using a clustered index is that only queries that benefit are the ones that use a highly selective filter on the clustered index column. To improve the filter on other columns, a nonclustered index can be added to other columns. However, each index that is added to a table adds both space and processing time to loads.

3) Heap Table - Non-index option (allow secondary index, no compression)

If we want to create a staging table in our dedicated SQL pool for loading data and transferring it, we will have to create a **Heap table**. Heap tables are faster to load and subsequent reads can be done from the cache. For small lookup tables, with less than 60 million rows, consider using HEAP or clustered index for faster query performance.

▼ Code Example

```
-- Creating a heap table

CREATE TABLE [dbo].[SalesFact_staging](
    [ProductID] [int] NOT NULL,
    [SalesOrderID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
```

```

[OrderQty] [smallint] NOT NULL,
[UnitPrice] [money] NOT NULL,
[OrderDate] [datetime] NULL,
[TaxAmt] [money] NULL
)
WITH(HEAP,
DISTRIBUTION = ROUND_ROBIN --Usually used for fast loading of data
)

CREATE INDEX ProductIDIndex ON [dbo].[SalesFact_staging] (ProductID)
--Explicitly create a non-clustered index since we are not using clustered table

```

Type	Great fit for...	Watch out if...
Heap	* Staging/temporary table * Small tables with small lookups	* Any lookup scans the full table
Clustered index	* Tables with up to 100 million rows * Large tables (more than 100 million rows) with only 1-2 columns heavily used	* Used on a replicated table * You have complex queries involving multiple join and Group By operations * You make updates on the indexed columns: it takes memory
Clustered columnstore index (CCI) (default)	* Large tables (more than 100 million rows)	* Used on a replicated table * You make massive update operations on your table * You overpartition your table: row groups do not span across different distribution nodes and partitions

/*If you intend to use a snowflake schema in which dimension tables are related to one another, you should include the key for the parent dimension in the definition of the child dimension table. For example, the following SQL code could be used to move the geographical address details from the DimCustomer table to a separate DimGeography dimension table:*/

```

CREATE TABLE dbo.DimGeography
(
    GeographyKey INT IDENTITY NOT NULL,
    GeographyAlternateKey NVARCHAR(10) NULL,
    StreetAddress NVARCHAR(100),
    City NVARCHAR(20),
    PostalCode NVARCHAR(10),
    CountryRegion NVARCHAR(20)
)
WITH
(

```



```

DISTRIBUTION = REPLICATE,
CLUSTERED COLUMNSTORE INDEX
);

CREATE TABLE dbo.DimCustomer
(
    CustomerKey INT IDENTITY NOT NULL,
    CustomerAlternateKey NVARCHAR(15) NULL,
    GeographyKey INT NULL,
    CustomerName NVARCHAR(80) NOT NULL,
    EmailAddress NVARCHAR(50) NULL,
    Phone NVARCHAR(25) NULL
)
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
);

/*The following code example creates a hypothetical fact table named FactSales that is r
elated to multiple dimensions through key columns (date, customer, product, and store)*/

CREATE TABLE dbo.FactSales
(
    OrderDateKey INT NOT NULL,
    CustomerKey INT NOT NULL,
    ProductKey INT NOT NULL,
    StoreKey INT NOT NULL,
    OrderNumber NVARCHAR(10) NOT NULL,
    OrderLineItem INT NOT NULL,
    OrderQuantity SMALLINT NOT NULL,
    UnitPrice DECIMAL NOT NULL,
    Discount DECIMAL NOT NULL,
    Tax DECIMAL NOT NULL,
    SalesAmount DECIMAL NOT NULL
)
WITH
(
    DISTRIBUTION = HASH(OrderNumber),
    CLUSTERED COLUMNSTORE INDEX
);

/*Staging tables are used as temporary storage for data as it's being loaded into the da
ta warehouse. The following code example creates a staging table for product data that w
ill ultimately be loaded into a dimension table:*/

CREATE TABLE dbo.StageProduct
(
    ProductID NVARCHAR(10) NOT NULL,
    ProductName NVARCHAR(200) NOT NULL,
    ProductCategory NVARCHAR(200) NOT NULL,
    Color NVARCHAR(10),
    Size NVARCHAR(10),
    ListPrice DECIMAL NOT NULL,
    Discontinued BIT NOT NULL

```

```

)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
);

/*In some cases, if the data to be loaded is in files with an appropriate structure, it
can be more effective to create external tables that reference the file location. This
way, the data can be read directly from the source files instead of being loaded into t
he relational store.*/

-- External data source links to data lake location
CREATE EXTERNAL DATA SOURCE StagedFiles
WITH (
    LOCATION = 'https://mydatalake.blob.core.windows.net/data/stagedfiles/'
);
GO

-- External format specifies file format
CREATE EXTERNAL FILE FORMAT ParquetFormat
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);
GO

-- External table references files in external data source
CREATE EXTERNAL TABLE dbo.ExternalStageProduct
(
    ProductID NVARCHAR(10) NOT NULL,
    ProductName NVARCHAR(200) NOT NULL,
    ProductCategory NVARCHAR(200) NOT NULL,
    Color NVARCHAR(10),
    Size NVARCHAR(10),
    ListPrice DECIMAL NOT NULL,
    Discontinued BIT NOT NULL
)
WITH
(
    DATA_SOURCE = StagedFiles,
    LOCATION = 'products/*.parquet',
    FILE_FORMAT = ParquetFormat
);

```

Table Partition

Logically splitting data into smaller manageable parts based on some column value e.g. splitting sales data by different provinces of Canada. **Normally, data is partitioned on the date column.**

As a result, partitioning helps is filtering data when using the WHERE clause in queries

Load Benefit: The primary benefit of partitioning in a dedicated SQL pool is to improve the efficiency and performance of loading data by use of partition deletion, switching, and merging. Partition switching can be used to quickly remove or replace a section of a table. Where deleting the individual rows could take hours, deleting an entire partition could take seconds.

Query Benefit: A query that applies a filter to partitioned data can limit the scan to only the qualifying partitions. This method of filtering can avoid a full table scan and only scan a smaller subset of data. With the introduction of clustered columnstore indexes, the predicate elimination performance benefits are less beneficial, but in some cases, there can be a benefit to queries.

▼ Code Example

```
-- Lab - Creating a table with partitions

DROP TABLE [logdata]

CREATE TABLE [logdata]
(
  [Id] [int] NULL,
  [Correlationid] [varchar](200) NULL,
  [Operationname] [varchar](200) NULL,
  [Status] [varchar](100) NULL,
  [Eventcategory] [varchar](100) NULL,
  [Level] [varchar](100) NULL,
  [Time] [datetime] NULL,
  [Subscription] [varchar](200) NULL,
  [Eventinitiatedby] [varchar](1000) NULL,
  [Resourcetype] [varchar](1000) NULL,
  [Resourcegroup] [varchar](1000) NULL
)

COPY INTO logdata FROM 'https://datalake2000.blob.core.windows.net/data/cleaned/Log.csv' WITH( FIRSTROW=2)

-- Let's first inspect our table to see the range of dates

SELECT FORMAT(Time,'yyyy-MM-dd') AS dt,COUNT(*) FROM logdata
GROUP BY FORMAT(Time,'yyyy-MM-dd')

/* Output will be partitioned by the date column For eg:
dt          (No column name)  ...
2021-01-01      501      ...
2021-01-02      45       ...
```

```

.. 2021-01-31      240      ...
.. 2021-12-01      11      ... and so on
We can see that we don't have an even distribution of data across the dates*/

-- Let's drop the existing table if it exists
DROP TABLE logdata

-- Let's create a new table with partitions
CREATE TABLE [logdata]
(
    [Id] [int] NULL,
    [Correlationid] [varchar](200) NULL,
    [Operationname] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Eventcategory] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Eventinitiatedby] [varchar](1000) NULL,
    [Resourcetype] [varchar](1000) NULL,
    [Resourcegroup] [varchar](1000) NULL
)
WITH
(
    PARTITION ( [Time] RANGE RIGHT FOR VALUES
        ('2021-04-01', '2021-05-01', '2021-06-01')
    )
)
/* First partition is any data <1st April,
   2nd partition between 1st Apr and 1st may,
   3rd partition between 1st May and 1st June
   4th any date >= 1st June */

-- Copy data into the table
COPY INTO logdata FROM 'https://datalake2000.blob.core.windows.net/data/cleaned/Log.csv' WITH ( FIRSTROW=2 )

-- View the partitions
SELECT QUOTENAME(s.[name])+'. '+QUOTENAME(t.[name]) as Table_name
,      i.[name] as Index_name
,      p.partition_number as Partition_nmbr
,      p.[rows] as Row_count
,      p.[data_compression_desc] as Data_Compression_desc
FROM    sys.partitions p
JOIN    sys.tables      t      ON    p.[object_id] = t.[object_id]
JOIN    sys.schemas    s      ON    t.[schema_id] = s.[schema_id]
JOIN    sys.indexes     i      ON    p.[object_id] = i.[object_id]
                                AND  p.[index_id]  = i.[index_id]
WHERE t.[name] = 'logdata'

```

```

/* The Output is:
Table_name      ... Partition_nmbr Row_count Data_compression_desc
[dbo].[logdata] ...          1         250      COLUMNSTORE
[dbo].[logdata] ...          2         250      COLUMNSTORE
[dbo].[logdata] ...          3         250      COLUMNSTORE
[dbo].[logdata] ...          4         250      COLUMNSTORE */

```

Partition Switching

```

-- Lab - Switching partitions

-- Create a new table with partitions
-- Switch partitions
-- This can be done with the Alter command.
-- But the alter command will not work if the table has a clustered column store in
dex
-- When using the CREATE TABLE AS, we need to mention a distribution type

CREATE TABLE [logdata_new]
WITH
(
DISTRIBUTION = ROUND_ROBIN, --Default option
PARTITION ( [Time] RANGE RIGHT FOR VALUES
            ('2021-05-01','2021-06-01')
          ) )
AS
SELECT *
FROM logdata
WHERE 1=2 --Since its always false, it will just copy the schema of the logdata to
the logdata_new

-- Switch the partitions and then see the data

ALTER TABLE [logdata] SWITCH PARTITION 2 TO [logdata_new] PARTITION 1;
--Data from portion 2 is going to be moved only and the previous logdata table will
be left with partitions 1,3,4

SELECT count(*) FROM [logdata_new]
SELECT FORMAT(Time,'yyyy-MM-dd') AS dt,COUNT(*) FROM logdata_new
GROUP BY FORMAT(Time,'yyyy-MM-dd')

```

Slowly Changing Dimensions (SCD)

Slowly changing dimensions (SCD) are tables in a dimensional model that handle changes to dimension values over time and not on a set schedule.

Over time, it is possible that certain product name changes or maybe a customer changes phone number. This will lead to the case where we will have to change the dimension table to reflect these changes. There are various strategies to tackle the different cases.

- **Type 1**

A Type 1 SCD always reflects the latest values, and when changes in source data are detected, the dimension table data is overwritten

E.g. When a customer's email address or phone number changes, the dimension table updates the customer row with the new values.



CustomerID	FirstName	LastName	EmailAddress	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	donna0@aw.com	A Bike Store	2021-03-20	2021-03-20

CustomerID	FirstName	LastName	EmailAddress	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	donna0@aw.com	Bikes, Bikes, Bikes	2021-03-20	2021-03-22

[Link to an example showing the detailed implementation steps](#)

- **Type 2 (important):**

A Type 2 SCD supports the versioning of dimension members. It includes columns that define the date range validity of the version (for example, `StartDate` and `EndDate`) and possibly a flag column (for example, `IsCurrent`) to easily filter by current dimension members.

Current versions may define an empty end date (or 12/31/9999), which indicates that the row is the current version. The table must also define a **surrogate key** because the business key (in this instance, employee ID) won't be unique.

SalesRepID	RepSourceId	FirstName	LastName	Region	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southcentral	2021-03-20	9999-12-31	True

SalesRepID	RepSourceId	FirstName	LastName	Region	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southcentral	2021-03-20	2021-03-21	False
3	331	Susan	Eaton	Southeast	2021-03-22	9999-12-31	True



NOTE: Surrogate keys are secondary row identification keys. They are added in all SCD2 cases because the primary identification key will not be unique anymore with newly added rows.

▼ Code to apply type 1 and 2 logic

```

/*Logic to implement Type 1 and Type 2 updates can be complex, and there are various techniques you can use. For example, you could use a combination of UPDATE and INSERT statements as shown in the following code example:*/

-- Insert new customers
INSERT INTO dbo.DimCustomer
SELECT stg.CustomerNo,
       stg.CustomerName,
       stg.EmailAddress,
       stg.Phone,
       stg.StreetAddress
FROM dbo.StageCustomers AS stg
WHERE NOT EXISTS
      (SELECT * FROM dbo.DimCustomer AS dim
       WHERE dim.CustomerAltKey = stg.CustomerNo);

-- Type 1 updates (name, email, phone)
UPDATE dbo.DimCustomer
SET CustomerName = stg.CustomerName,
    EmailAddress = stg.EmailAddress,
    Phone = stg.Phone
FROM dbo.StageCustomers AS stg
WHERE dbo.DimCustomer.CustomerAltKey = stg.CustomerNo;

```

```

-- Type 2 updates (geographic address)
INSERT INTO dbo.DimCustomer
SELECT stg.CustomerNo AS CustomerAltKey,
       stg.CustomerName,
       stg.EmailAddress,
       stg.Phone,
       stg.StreetAddress,
       stg.City,
       stg.PostalCode,
       stg.CountryRegion
FROM dbo.StageCustomers AS stg
JOIN dbo.DimCustomer AS dim
ON stg.CustomerNo = dim.CustomerAltKey
AND stg.StreetAddress <> dim.StreetAddress;

/*As an alternative to using multiple INSERT and UPDATE statement, you can use a
single MERGE statement to perform an "upsert" operation to insert new records an
d update existing ones, as shown in the following example, which loads new produ
ct records and applies type 1 updates to existing products*/

MERGE dbo.DimProduct AS tgt
      USING (SELECT * FROM dbo.StageProducts) AS src
      ON src.ProductID = tgt.ProductBusinessKey
WHEN MATCHED THEN
      UPDATE SET
          tgt.ProductName = src.ProductName,
          tgt.ProductCategory = src.ProductCategory,
          tgt.Color = src.Color,
          tgt.Size = src.Size,
          tgt.ListPrice = src.ListPrice,
          tgt.Discontinued = src.Discontinued
WHEN NOT MATCHED THEN
      INSERT VALUES
          (src.ProductID,
           src.ProductName,
           src.ProductCategory,
           src.Color,
           src.Size,
           src.ListPrice,
           src.Discontinued);

/*Another way to load a combination of new and updated data into a dimension tab
le is to use a CREATE TABLE AS (CTAS) statement to create a new table that conta
ins the existing rows from the dimension table and the new and updated records f
rom the staging table. After creating the new table, you can delete or rename th
e current dimension table, and rename the new table to replace it.*/

CREATE TABLE dbo.DimProductUpsert
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
-- New or updated rows

```



```

SELECT  stg.ProductID AS ProductBusinessKey,
        stg.ProductName,
        stg.ProductCategory,
        stg.Color,
        stg.Size,
        stg.ListPrice,
        stg.Discontinued
FROM    dbo.StageProduct AS stg
UNION ALL
-- Existing rows
SELECT  dim.ProductBusinessKey,
        dim.ProductName,
        dim.ProductCategory,
        dim.Color,
        dim.Size,
        dim.ListPrice,
        dim.Discontinued
FROM    dbo.DimProduct AS dim
WHERE NOT EXISTS
(
  SELECT *
  FROM  dbo.StageProduct AS stg
  WHERE stg.ProductId = dim.ProductBusinessKey
);

RENAME OBJECT dbo.DimProduct TO DimProductArchive;
RENAME OBJECT dbo.DimProductUpsert TO DimProduct;

```

- **Type 3:**

A Type 3 SCD supports storing two versions of a dimension member as separate columns.

Here instead of having multiple rows to signify changes, we have multiple columns. We do have an effective/modified date column to show when the change took place.

CustomerID	FirstName	LastName	CurrentEmail	OriginalEmail	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	donna0@aw.com	donna0@aw.com	A Bike Store	2021-03-20	2021-03-20
CustomerID	FirstName	LastName	CurrentEmail	OriginalEmail	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	dc3@aw.com	donna0@aw.com	A Bike Store	2021-03-20	2021-03-22

- **Type 6:**

A Type 6 SCD combines Type 1, 2, and 3. In Type 6 design we also store the current value in all versions of that entity so you can easily report the current value or the historical value.

SalesRepID	RepSourceId	FirstName	LastName	CurrentRegion	HistoricalRegion	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southcentral	Southcentral	2021-03-20	9999-12-31	True

SalesRepID	RepSourceId	FirstName	LastName	CurrentRegion	HistoricalRegion	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southeast	Southcentral	2021-03-20	2021-03-21	False
3	331	Susan	Eaton	Southeast	Southeast	2021-03-22	9999-12-31	True

Window Functions (also used in stream analytics)

A window function enables you to perform a mathematical equation on a set of data that is defined within a window. The mathematical equation is typically an aggregate function; however, instead of applying the aggregate function to all the rows in a table, it is applied to a set of rows that are defined by the window function, and then the aggregate is applied to it.

A suite of functions that are helpful and easier compared to complex queries since they reduce the need for intermediate tables to store temporary data. These functions are used only when we have a requirement to work with a specific window/time period.

For e.g.

Running Total of Revenue for each Week,

Top N products for a week's sale,

Moving Averages over the last 3 rows

When using the windowing function with SQL pools, we will use the **OVER** clause. This clause determines the partitioning and ordering of a rowset before the window function is applied.

▼ Code Example

```

-- Lab - Windowing Functions

SELECT
ROW_NUMBER() OVER(
    PARTITION BY [ProductID]
    ORDER BY [OrderQty]) AS "Row Number"
,[ProductID]
,[CustomerID]
,[OrderQty]
,[UnitPrice]
FROM [dbo].[SalesFact]
ORDER BY [ProductID]

/* Output will be partitioned by the Product Id column For eg:
   Row num   ProdId   ...   ...   ...
       1       700     ...
       2       700     ...
..  12       700     ...
       1       701     ... and so on*/

SELECT
ROW_NUMBER() OVER(
    PARTITION BY [ProductID]
    ORDER BY [OrderQty]) AS "Row Number"
,[ProductID]
,[CustomerID]
,[OrderQty]
,SUM([OrderQty]) OVER(
    PARTITION BY [ProductID]) AS TotalOrderQty
,[UnitPrice]
FROM [dbo].[SalesFact]
ORDER BY [ProductID]

#Running Sum - Order by day
FROM groceries
SELECT id
, revenue
, day
,SUM(revenue) over (order by day) as running_total;

```

Surrogate Keys

Generally, data for dimension tables can come from multiple sources and if the primary key column for these tables is the same then we won't have a way to distinguish between the different rows. Thus we would want to have **surrogate keys**

to distinguish the rows (*as simple as a new index column*). The surrogate key is also referred to as a **non-business key**

▼ Code Example

```
-- Lab - Surrogate keys for dimension tables

-- First let's ensure we have the tables defined in the SQL pool
-- Let's do this for one dimension table

-- First drop the table if you have it in place

DROP TABLE [dbo].[DimProduct]

CREATE TABLE [dbo].[DimProduct](
  [ProductSK] [int] IDENTITY(1,1) NOT NULL, --destination table surrogate key column
  --the values will be generated automatically but not necessarily in incrementing fashion
  [ProductID] [int] NOT NULL,
  [ProductModelID] [int] NOT NULL,
  [ProductSubcategoryID] [int] NOT NULL,
  [ProductName] varchar(50) NOT NULL,
  [SafetyStockLevel] [smallint] NOT NULL,
  [ProductModelName] varchar(50) NULL,
  [ProductSubCategoryName] varchar(50) NULL
)
```

✓ Dynamic data masking

***Dynamic data masking** helps prevent unauthorized access to sensitive data by enabling customers to designate how much of the sensitive data to reveal with minimal impact on the application layer. It's a policy-based security feature that hides the sensitive data in the result set of a query over designated database fields, while the data in the database is not changed.*

- **No change in the physical layer**
 - Data in the database is not changed
 - Not the same as data encryption

- **No additional development effort is needed at the application level**
- **Security: Should not be used as a primary security layer**
 - DDM should not be used as an isolated measure to fully secure sensitive data
 - ad-hoc query permissions can apply techniques to gain access to the actual data.
- **Other considerations**
 - Masked columns can be updated if the user has permission
 - Export masked from source data results in masked data in the target table

ID	PersonName	EmailAddress	CreditCardNumber	SocialSecurityNumber
1	Anoop Kumar	abcdefg@hotmai.com	1234-5678-4321-8765	123-45-6789
1	Rahul Gupta	amitguptaabcdefg@hotmai.com	8765-1234-5678-4321	231-45-6787
1	Amit Goel	amitgoelabcdefgh@hotmai.com	4321-1234-5678-4321	321-45-6700

ID	PersonName	EmailAddress	CreditCardNumber	SocialSecurityNumber
9590	AXXXr	aXXX@XXXX.com	xxxx-xxxx-xxxx-8765	xxxx
7604	RXXXa	aXXX@XXXX.com	xxxx-xxxx-xxxx-4321	xxxx
8453	AXXXI	aXXX@XXXX.com	xxxx-xxxx-xxxx-4321	xxxx

Random Number function – generates random number based on selected boundaries and actual data type. Can be applied only numbers, not string

Email function – Exposes the first letter and replace the domain with xxx.com

Default function – Full masking of data. For numeric – 0 For String – XXXX characters

Custom String function – You can define the exposed prefix, the padding string and exposed suffix

Credit Card function - Only the last four digits of Credit card are shown

Function	Description	Examples
Default	Full masking according to the data types of the designated fields. For string data types, use XXXX or fewer Xs if the size of the field is fewer than 4 characters. For numeric data types use a zero value. For date and time data types use 01.01.1900 00:00:00.0000000. For binary data types use a single byte of ASCII value 0.	Example column definition syntax: <code>Phone# varchar(12) MASKED WITH (FUNCTION = 'default()') NULL</code>
Email	Masking method that exposes the first letter of an email address and the constant suffix ".com", in the form of an email address <code>aXXX@XXXX.com</code> .	Example definition syntax: <code>Email varchar(100) MASKED WITH (FUNCTION = 'email()') NULL</code>

Function	Description	Examples
Random	A random masking function for use on any numeric type to mask the original value with a random value within a specified range.	Example definition syntax: <code>Account_Number bigint MASKED WITH (FUNCTION = 'random([start range], [end range]))'</code>
Custom String	Masking method that exposes the first and last letters and adds a custom padding string in the middle. <code>prefix, [padding], suffix</code> Note: If the original value is too short to complete the entire mask, part of the prefix or suffix won't be exposed.	Example definition syntax: <code>FirstName varchar(100) MASKED WITH (FUNCTION = 'partial(prefix, [padding],suffix)') NULL</code>

▼ Code Example

```
-- Azure Example
--DROP TABLE TestDDM
Create table TestDDM
    (ID Int,
    PersonName varchar (100),
    EmailAddress varchar(120),
    CreditCardNumber varchar(19),
    SocialSecurityNumber varchar(11)
)

INSERT INTO TestDDM Values (1, 'Anoop Kumar','abcdefgh@hotmail.com','1234-5678-432
1-8765','123-45-6789')
INSERT INTO TestDDM Values (1, 'Rahul Gupta','amitguptaabcdefgh@hotmail.com','8765-
1234-5678-4321','231-45-6787')
INSERT INTO TestDDM Values (1, 'Amit Goel','amitgoelabcdefgh@hotmail.com','4321-12
34-5678-4321','321-45-6700')

SELECT * FROM TestDDM

/* After this we can go into Azure -> Security -> Dynamic Data masking where we can
provide all the functions. Similar task can be done from SQL query as well */

-- SQL Server Example

CREATE TABLE Membership
(MemberID int IDENTITY PRIMARY KEY,
FirstName varchar(100) MASKED WITH (FUNCTION = 'partial(1,"XXXXXXX",0)') NULL,
LastName varchar(100) NOT NULL,
Phone varchar(12) MASKED WITH (FUNCTION = 'default()') NULL,
Email varchar(100) MASKED WITH (FUNCTION = 'email()') NULL);
```

Search (Ctrl+/) << Save Discard **+ Add mask** Feedback

Learn more - Getting Started Guide

Masking rules

Mask name	Mask Function
You haven't created any masking rules.	

SQL users excluded from masking (administrators are always excluded) ⓘ

SQL users excluded from masking (administrators are always excluded) ✓

Recommended fields to mask

Schema	Table	Column	
dbo	EmailAnalytics	Zip_Code	Add mask
dbo	EmailAnalytics	Email_Status	Add mask
dbo	department_visit_cust...	Phone_and_GPS	Add mask
dbo	CustomerVisitF_Spark	Phone_and_GPS	Add mask
wwi_poc	Customer	FirstName	Add mask
wwi_poc	Customer	LastName	Add mask
wwi_poc	Customer	FullName	Add mask
wwi_poc	Customer	BirthDate	Add mask
wwi_poc	Customer	Address_PostalCode	Add mask

Navigation menu:

- Overview
- Activity log
- Access control (IAM)
- Tags
- Settings
 - Workload management
 - Maintenance schedule
 - Geo-backup policy
 - Connection strings
 - Properties
 - Locks
- Security
 - Auditing
 - Data Discovery & Classification
 - Dynamic Data Masking**
 - Security Center
 - Transparent data encryption

✓ Workload management

Workload management is the process of allowing administrators to control certain aspects of the warehouse to perform at optimal levels when executing tasks such as loading and transforming data.

Dedicated SQL pool workload management in Azure Synapse consists of three high-level concepts:

- **Workload Classification**

Workload management classification allows workload policies to be applied to requests through assigning resource classes and importance.

The simplest and most common classification is load and query. For eg, having a workload policy for load activity assigning it a higher resource class with more

resources and another workload policy for querying, providing it lower importance compared to load activities.

- **Workload Importance**

Workload importance influences the order in which a request gets access to resources. On a busy system, a request with higher importance has first access to resources. There are five levels of importance: low, below_normal, normal, above_normal, and high. Requests that don't set importance are assigned the default level of normal.

- **Workload Isolation**

Workload isolation reserves resources for a workload group. Resources reserved in a workload group are held exclusively for that workload group to ensure execution. Workload groups give you the ability to reserve or cap the amount of resources a set of requests can consume. Finally, workload groups are a mechanism to apply rules, such as query timeout, to requests.

```
/* We can create multiple workload groups in order to provision compute resources such that two different tasks such that an user loading data doesn't use the full resource capacity when an user already performing some analysis job*/

CREATE WORKLOAD GROUP DataLoads --Workload Isolation
WITH (
    MIN_PERCENTAGE_RESOURCE = 80
    ,CAP_PERCENTAGE_RESOURCE = 100
    ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 4 -- factor of 80 (guaranteed more than 20 concurrencies)
);
--[Max Concurrency] = [CAP_PERCENTAGE_RESOURCE] / [REQUEST_MIN_RESOURCE_GRANT_PERCENT]

CREATE WORKLOAD CLASSIFIER [ELTLogin] --Workload Classification
WITH (
    WORKLOAD_GROUP = 'DataLoads'
    ,MEMBERNAME = 'user_load'
    ,IMPORTANCE = High -- Workload Importance
);
```

Materialized Views

Views are logical projections of data from multiple tables. A standard view computes its data each time when the view is used. There's no data stored on disk. **A materialized view pre-computes, stores, and maintains its data in a dedicated**

SQL pool just like a table. They are not supported by default in serverless SQL pools. Recomputation isn't needed each time a materialized view is used. That's why queries that use all or a subset of the data in materialized views can gain faster performance.

Comparison	View	Materialized View
View definition	Stored in Azure data warehouse.	Stored in Azure data warehouse.
View content	Generated each time when the view is used.	Pre-processed and stored in Azure data warehouse during view creation. Updated as data is added to the underlying tables.
Data refresh	Always updated	Always updated
Speed to retrieve view data from complex queries	Slow	Fast
Extra storage	No	Yes
Syntax	CREATE VIEW	CREATE MATERIALIZED VIEW AS SELECT

Materialized views results in increased performance since the data within the view can be fetched without having to resolve the underlying query to base tables. You can also further filter and supplement other queries as if it is a table also. In addition, you also can define a different table distribution within the materialized view definition that is different from the table on which it is based. As the data in the underlying base tables change, the data in the materialized view will automatically update without user interaction.

There are several restrictions that you must be aware of before defining a materialized view:

- The SELECT list in the materialized view definition needs to meet at least one of these two criteria:
 - The SELECT list contains an aggregate function.
 - GROUP BY is used in the Materialized view definition and all columns in GROUP BY are included in the SELECT list. Up to 32 columns can be used in the GROUP BY clause.
- Supported aggregations include MAX, MIN, AVG, COUNT, COUNT_BIG, SUM, VAR, STDEV.

- Only the hash and round_robin table distribution is supported in the definition.
- Only CLUSTERED COLUMNSTORE INDEX is supported by materialized view.

```
CREATE MATERIALIZED VIEW [ schema_name. ] materialized_view_name
  WITH (
    <distribution_option>
  )
  AS <select_statement>
[;]

<distribution_option> ::=
{
  DISTRIBUTION = HASH ( distribution_column_name )
| DISTRIBUTION = HASH ( [distribution_column_name [, ...n]] )
| DISTRIBUTION = ROUND_ROBIN
}

<select_statement> ::=
  SELECT select_criteria
```

```
--Example
--When MIN/MAX aggregates are used in the SELECT list of materialized view definition, F
OR_APPEND is required

CREATE MATERIALIZED VIEW mv_test2
WITH (distribution = hash(i_category_id), FOR_APPEND)
AS
SELECT MAX(i.i_rec_start_date) as max_i_rec_start_date, MIN(i.i_rec_end_date) as min_i_r
ec_end_date, i.i_item_sk, i.i_item_id, i.i_category_id
FROM syntheticworkload.item i
GROUP BY i.i_item_sk, i.i_item_id, i.i_category_id
```

✓ Result set Caching

Caching refers to storing intermediate data in faster storage layers to speed up queries. When result set caching is enabled, a dedicated SQL pool **automatically caches query results** in the user database for repetitive use. Thus, **enable resultset caching when you expect results from queries to return the same values.**

This option stores a copy of the result set on the control node so that queries do not need to pull data from the storage subsystem or compute nodes. The capacity for the resultset cache is 1 TB and the data within the resultset cache is expired and purged after 48 hours of not being accessed.

Azure Synapse SQL automatically caches query results in the user database for repetitive use. Resultset caching allows subsequent query executions to get results directly from the persisted cache so recomputation is not needed. Result set caching improves query performance and reduces compute resource usage.

To enable result set caching, run this command when connecting to the MASTER database.

```
ALTER DATABASE [database_name]
SET RESULT_SET_CACHING ON;
```

✓ Row and Column Level Security

Column-level security simplifies the design and coding of security in your application, allowing you to restrict column access to protect sensitive data. For example, ensuring those specific users can access only certain columns of a table pertinent to their department. The way to implement column-level security is by using the **GRANT** T-SQL statement.

```
GRANT SELECT on dbo.Customers
(CustomerId, FirstName, LastName) to
ContractEmp
```

```
GRANT <permission> [ ,...n ] ON
  [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ] // specifying the
column access
  TO <database_principal> [ ,...n ]
```

Row-level security (RLS) can help you to create a group membership or execution context in order to control not just columns in a database table, but actually, the rows. The way to implement RLS is by using the **CREATE SECURITY POLICY** statement. For reading [more](#).

✓ Transparent Data Encryption

Transparent data encryption (TDE) helps protect Azure Synapse Analytics against the threat of malicious offline activity by encrypting data at rest. It performs real-time encryption and decryption of the database, associated backups, and transaction log files at rest without requiring changes to the application. It is enabled by default.

Managed identities provide Azure services with an automatically managed identity in Azure Active Directory. You can use the Managed Identity capability to authenticate to any service that supports Azure Active Directory authentication.

✓ Statistics to improve query performance

When queries are submitted, a dedicated SQL pool query optimizer tries to determine which access paths to the data will result in the least amount of effort to retrieve the data required to resolve the query. It is a cost-based optimizer that compares the cost of various query plans and then chooses the plan with the lowest cost. After loading data into a dedicated SQL pool, collecting statistics on your data is one of the most important things you can do for query optimization.

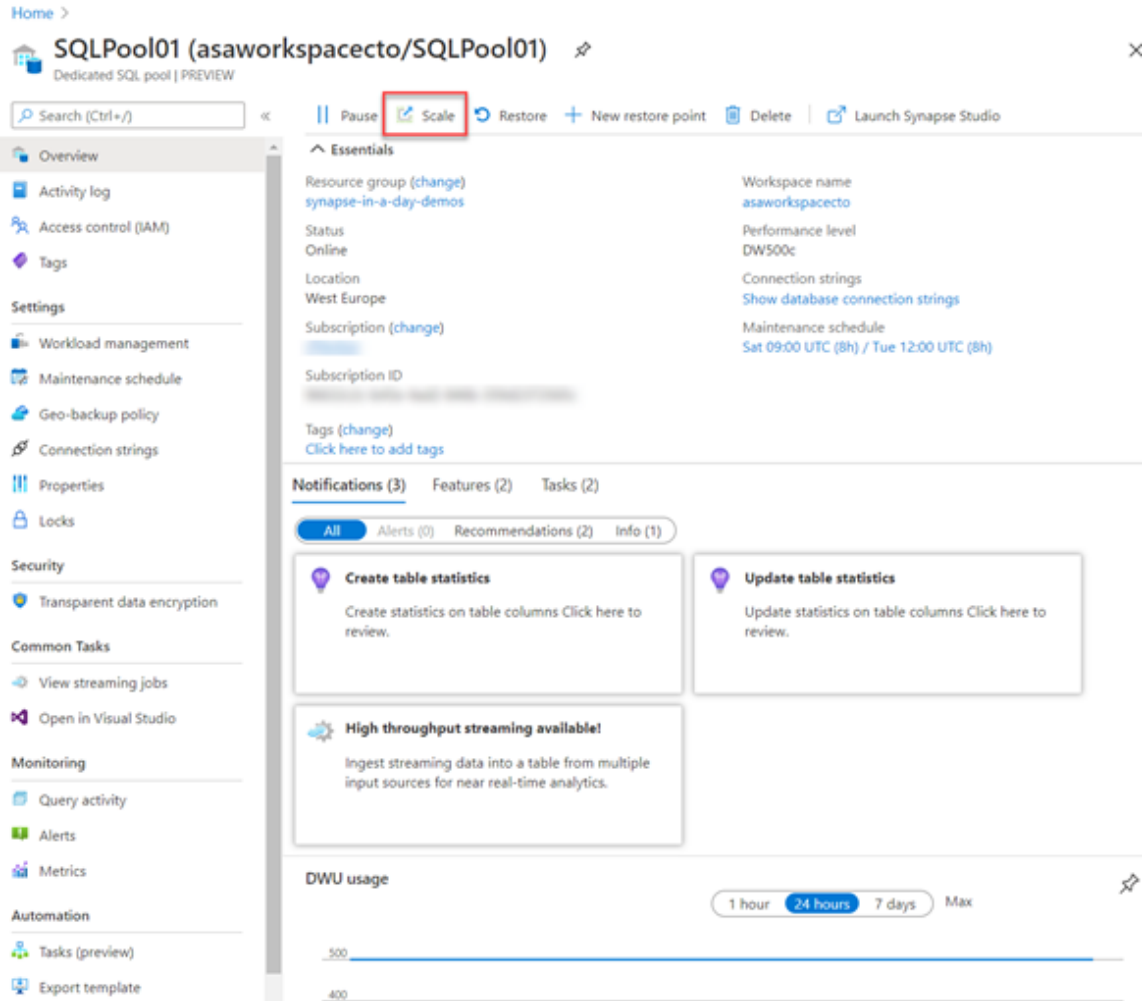
It tracks cardinality and range density to determine which data access paths return the fewest rows for speed.

For example, if the optimizer estimates that the date your query is filtering on will return one row, it will choose one plan. If it estimates that the selected date will return 1 million rows, it will return a different plan.

✓ Scale Compute Resources

In **SQL pools**, the unit of scale is an abstraction of compute power that is known as a data warehouse unit. Compute is separate from storage, which enables you to scale compute independently of the data in your system. This means you can scale up and scale down the compute power to meet your needs.

You can scale a Synapse SQL pool either through the Azure portal, Azure Synapse Studio or programmatically using TSQL or PowerShell.



Scale

SQLPool01

Configure the settings that best align to the workload needs on the dedicated SQL pool. [Learn more about performance levels](#)

Performance level



Estimated price

Est. cost per hour

▼ DATA EXPLORER (optional)

If you want to learn more about this, click [here](#).

▼ **APACHE SPARK (Important)**

Apache Spark processes large amounts of data **in-memory**, which boosts the performance of analyzing big data more effectively, and this capability is available within Azure Synapse Analytics referred to as **Spark pools**.

Spark pool clusters are groups of computers that are treated as a single computer and handle the execution of commands issued from notebooks. The clusters allow the processing of data to be parallelized across many computers to improve scale and performance. It consists of a Spark Driver and Worker nodes. **Spark pools in Azure Synapse can use Azure Data Lake Storage Generation 2 as well as BLOB storage.**

The primary use case for Apache Spark for Azure Synapse Analytics is to process big data workloads that cannot be handled by Azure Synapse SQL, and where you don't have an existing Apache Spark implementation.

There are two ways within Synapse to use Spark:

- **Spark Notebooks** for doing Data Science and Engineering use Scala, PySpark, C#, and SparkSQL
- **Spark job definitions** for running batch Spark jobs using jar files.

[Link to an example showing the detailed implementation steps](#)

Indexing

In Azure, we have technologies that can perform indexing on huge volumes of data. These indexes can then be used by analytical engines such as Spark to speed up the queries. One such technology that Azure offers is called **Hyperspace**.

Hyperspace lets us create indexes on input datasets such as Parquet, CSV, and so on, which can be used for query optimization. The Hyperspace indexing needs to be run separately to create an initial index. After that, it can be incrementally updated for the new data. Once we have the Hyperspace index, any Spark query can leverage the index, similar to how we use indexes in SQL.

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 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.

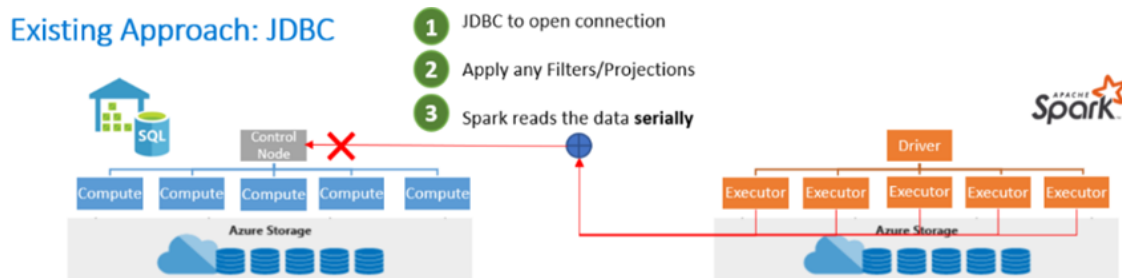
Delta Lake for Streaming Data

Integrate SQL and Apache Spark Pools

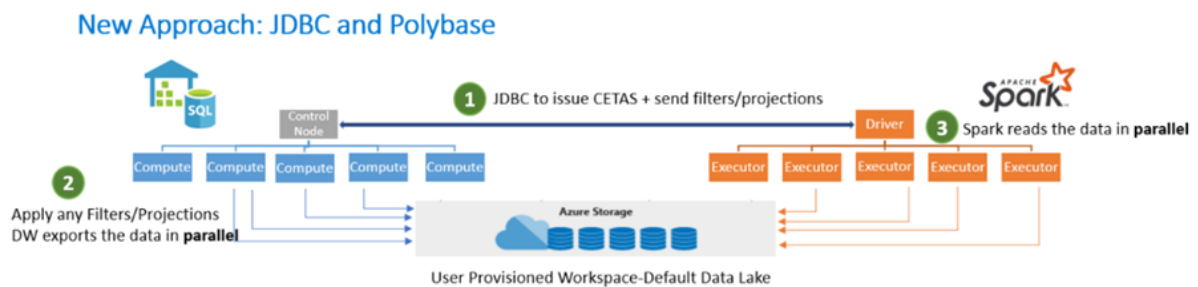
The **Apache Spark to Synapse SQL** connector is designed to efficiently transfer data between serverless Apache Spark pools and dedicated SQL pools in Azure Synapse. At the moment, the Azure Synapse Apache Spark to Synapse SQL connector works on dedicated SQL pools only, it doesn't work with serverless SQL pools.

The JDBC API opens the connection, filters, and applies projections, and Apache Spark reads the data serially. Given that two distributed systems such as Apache

Spark and SQL pools are being used, using the JDBC API becomes a bottleneck with a serial transfer of data.



Therefore, a new approach is to use both JDBC and PolyBase. First, the JDBC opens a connection, issues Create External Tables As Select (CETAS) statements, and sends filters and projections. The filters and projections are then applied to the data warehouse and exported in parallel using PolyBase. Apache Spark reads the data in parallel based on the user-provisioned workspace and the default data lake storage.



As a result, you can use the Azure Synapse Apache Spark Pool to Synapse SQL connector to transfer data between a Data Lake store via Apache Spark and dedicated SQL Pools efficiently.

When you deploy an Azure Synapse Apache Spark cluster, the Azure Data Lake Gen2 capability enables you to store Apache Spark SQL Tables within it. If you use Apache Spark SQL tables, these tables can be queried from a SQL-based Transact-SQL language without needing to use commands like CREATE EXTERNAL TABLE. Within Azure Synapse Analytics, these queries integrate natively with data files that are stored in an Apache Parquet format.

The integration can be helpful in use cases where you perform an ETL process predominately using SQL but need to call on the computation power of Apache Spark

to perform a portion of the extract, transform, and load (ETL) process as it is more efficient.

Authentication

The authentication between the two systems is made seamless in Azure Synapse Analytics. The Token Service connects with Azure Active Directory to obtain the security tokens to be used when accessing the storage account or the data warehouse in the dedicated SQL pool.

For this reason, there's no need to create credentials or specify them in the connector API if Azure AD-Auth is configured at the storage account and the dedicated SQL pool. If not, SQL Authentication can be specified. **The only constraint is that this connector only works in Scala.**

[Read more](#)

Monitor and Manage workloads

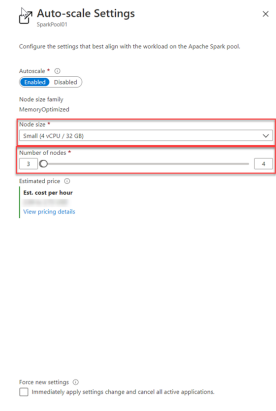
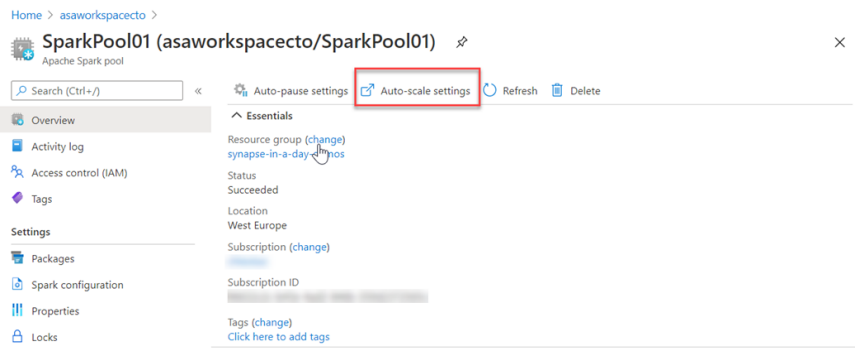
Scale Compute Resources

Apache Spark pools for Azure Synapse Analytics uses an **Autoscale** feature that automatically scales the number of nodes in a cluster instance up and down. During the creation of a new Spark pool, a minimum and maximum number of nodes can be set when **Autoscale** is selected. Autoscale then monitors the resource requirements of the load and scales the number of nodes up or down. To enable the Autoscale feature, complete the following steps as part of the normal pool creation process:

1. On the **Basics** tab, select the **Enable autoscale** checkbox.
2. Enter the desired values for the following properties:
 - **Min** number of nodes.
 - **Max** number of nodes.

The initial number of nodes will be the minimum. This value defines the initial size of the instance when it's created. The minimum number of nodes can't be fewer than three.

You can also modify this in the Azure portal, you can click on the **auto-scale settings** icon



Choose the node size
and the number of
nodes

▼ SYNAPSE LINK

Hybrid Transactional and Analytical Processing enables businesses to perform analytics over a database system that is seen to provide transactional capabilities without impacting the performance of the system. This enables organizations to use a database to fulfill both transactional and analytical needs to support near real-time analysis of operational data to make decisions about the information that is being analyzed.

In an HTAP solution, the transactional data is replicated automatically, with low latency, to an analytical store, where it can be queried without impacting the performance of the transactional system.

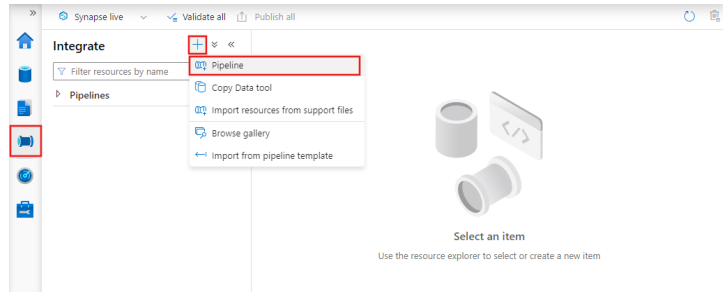
[Link to an example showing the detailed implementation steps](#)

Synapse Link for SQL - Learn more [here](#) or [here](#).

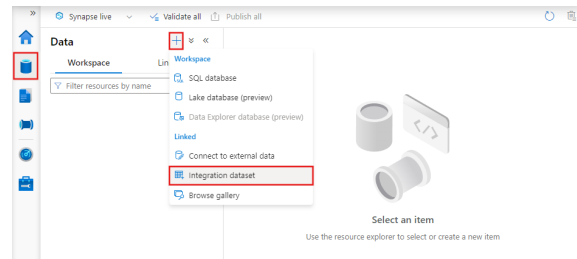
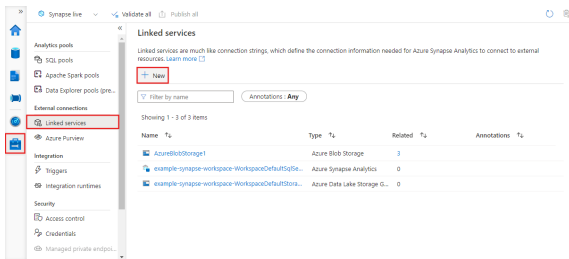
▼ PIPELINE AND DATA FLOW (Important)

Just like Azure Data Factory, Azure Synapse 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.

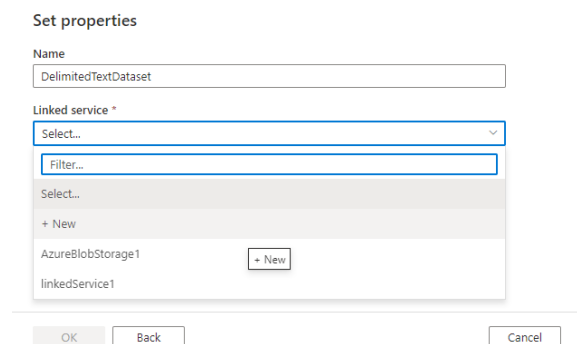
Azure Synapse Analytics just like ADF has the same three groupings of activities: data movement activities, data transformation activities, and control activities. For revising the concepts from ADF, click [here](#).



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 or Synapse Workspace. 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).



You can choose an existing linked service of the type you selected for the dataset, or create a new one if one isn't already defined.



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.

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 same triggers as in ADF: scheduled, tumbling window, and event-based.

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.

✓ So what are the differences between ADF and ASA ??

Category	Feature	Azure Data Factory	Azure Synapse Analytics
Integration Runtime	Using SSIS and SSIS Integration Runtime	✓	✓ <i>Public preview</i>
	Support for Cross-region Integration Runtime (Data Flows)	✓	✗
	Integration Runtime Sharing	✓ <i>Can be shared across different data factories</i>	✗
Pipelines Activities	SSIS Package Activity	✓	✓ <i>Public preview</i>
	Support for Power Query Activity (Wrangling Data Flow)	✓	✗
	Support for global parameters	✓	✗
GIT Repository Integration	GIT Integration	✓	✓
Monitoring	Monitoring of Spark Jobs for Data Flow	✗	✓ <i>Leverage the Synapse Spark pools</i>

✓ Data Flows

Since it has already been covered in ADF, please refer there using this [link](#).

✓ Loading Methods

Analytical systems are constantly balanced between loading and querying workloads. One of the main design goals in loading data is to manage or minimize the impact on analytical workloads while loading the data with the highest throughput possible.

Singleton updates: **Singleton** or smaller transaction batch loads should be grouped into larger batches to optimize the Synapse SQL Pools processing capabilities. One way to solve this issue is to develop one process that writes the outputs of an INSERT statement to a file and then another process to periodically load this file to take advantage of the parallelism.

Single Client loading method

- SSIS
- Azure Data Factory

Can add some parallel capabilities but are bottlenecked at the control node

Parallel Reader loading method

- PolyBase

Reads from Azure blob and loads to Azure Synapse Analytics, SQL Server etc

Bypasses control node and loads directly into Compute nodes



NOTE:

While dedicated SQL pools support many loading methods, including popular SQL Server options such as BCP and the SqlBulkCopy API, **the fastest and most scalable way to load data is through PolyBase external tables and the COPY INTO <table> FROM command.**

If you are using PolyBase, you need to define external tables in your dedicated SQL pool before loading. PolyBase uses external tables to define and access the data in Azure Storage. An external table is similar to a database view. The external table contains the table schema and points to data that is stored outside the dedicated SQL pool.

PolyBase can't load **rows** that have **more than 1MB** of data. When you put data into the text files in Azure Blob storage or Azure Data Lake Store, they must have fewer than 1,000,000 bytes of data. This byte limitation is true regardless of the table schema.

✓ 1) Load data to External Tables/ Serverless SQL Pool

The data lies in other data sources such as Hadoop, Azure Blob storage, or Azure Data lake Storage, whereas only the table structure is present in Azure Synapse. External tables are accessed using a feature called **PolyBase**

PolyBase is a tool that enables services such as SQL Server and Synapse Dedicated SQL pool to copy and query data directly from external locations. PolyBase is integrated into T-SQL, so every time we use a `COPY INTO <table> FROM` command to read data from an external storage location, PolyBase kicks in. PolyBase is one of the fastest and most scalable ways to copy data.

In order to access external tables:

Step 1 → Authorization to use the Data Lake storage account

Step 2 → Define the format of the external file that we will work with e.g. CSV, parquet, etc

Step 3 → Create and access the external table

▼ Code Examples

```
-- Lab - Using External tables (Serverless SQL Pool)

/* PolyBase 6 steps Process

1. CREATE DATABASE ENCRYPTION KEY
2. CREATE DATABASE-SCOPED CREDENTIAL
3. CREATE AN EXTERNAL DATA SOURCE
4. CREATE AN EXTERNAL FILE FORMAT
5. CREATE AN EXTERNAL TABLE
6. CREATE A TABLE AS */

-- First we need to create a database in the serverless pool
CREATE DATABASE [appdb]

/* Ensure to switch the context to the new database (appdb) first */

/* 1. To access your Data Lake Storage account, you will need to create a
Database Master Key to encrypt your credential secret. You then create a Database S
coped Credential to store your secret. The Master Key is required to encrypt the cr
edential secret (Shared Access Signature) in the next step. */

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd@123';

/* 2.(for blob storage key authentication): Create a database scoped credential
IDENTITY: Provide any string, it is not used for authentication to Azure storag
e.
SECRET: Provide your Azure storage account key (SAS).
*/
-- Here we are using the Shared Access Signature to authorize the use of the Azure
Data Lake Storage account

CREATE DATABASE SCOPED CREDENTIAL SasToken
WITH IDENTITY='SHARED ACCESS SIGNATURE'
, SECRET = 'sv=2020-02-10&ss=b&srt=sco&sp=rl&se=2021-06-26T14:34:27Z&st=2021-06-26T
06:34:27Z&spr=https&sig=7nxID0JFYuddBCnNTsPoeyY%2BRZokkcgdSUSsrfmAkRc%3D';
--this is the SAS token that will be generated from the Data Lake based on options

/* 3 (for blob): Create an external data source
TYPE: HADOOP - PolyBase uses Hadoop APIs to access data in Azure Data Lake Stora
ge.
LOCATION: Provide Data Lake Storage blob account name and URI
CREDENTIAL: Provide the credential created in the previous step.
*/

CREATE EXTERNAL DATA SOURCE log_data
```

```

WITH ( LOCATION = 'https://storageneil.dfs.core.windows.net/data',
        CREDENTIAL = SasToken
)

/* 4: Create an external file format
   FIELD_TERMINATOR: Marks the end of each field (column) in a delimited text file
   STRING_DELIMITER: Specifies the field terminator for data of type string in the
text-delimited file.
   DATE_FORMAT: Specifies a custom format for all date and time data that might app
ear in a delimited text file.
   Use_Type_Default: Store missing values as default for datatype.
*/

CREATE EXTERNAL FILE FORMAT TextFileFormat WITH (
    FORMAT_TYPE = DELIMITEDTEXT, --for CSV files
    FORMAT_OPTIONS (
        FIELD_TERMINATOR = ',',
        FIRST_ROW = 2))

/* 5: Create an External Table
   LOCATION: Folder under the Data Lake Storage root folder.
   DATA_SOURCE: Specifies which Data Source Object to use.
   FILE_FORMAT: Specifies which File Format Object to use
   REJECT_TYPE: Specifies how you want to deal with rejected rows. Either Value or
percentage of the total
   REJECT_VALUE: Sets the Reject value based on the reject type.
*/

/* IMP NOTE
External Tables are strongly typed.
This means that each row of the data being ingested must satisfy the table schema d
efinition. If a row does not match the schema definition, the row is rejected from
the load.
*/

CREATE EXTERNAL TABLE [logdata]
(
    [Id] [int] NULL,
    [Correlationid] [varchar](200) NULL,
    [Operationname] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Eventcategory] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Eventinitiatedby] [varchar](1000) NULL,
    [Resourcetype] [varchar](1000) NULL,
    [Resourcegroup] [varchar](1000) NULL)
WITH (
    LOCATION = '/Log.csv',

```



```

DATA_SOURCE = log_data, --Data source name defined above with SAS token
FILE_FORMAT = TextFileFormat
)

/* 6 CREATE TABLE AS - CTAS
   CTAS creates a new table and populates it with the results of a select statement.
   CTAS defines the new table to have the same columns and data types as the results
   of the select statement.
   If you select all the columns from an external table, the new table is a replica
   of the columns and data types in the external table.
*/

CREATE TABLE [EventHistory]
WITH (DISTRIBUTION = HASH([OperationName] ) )
AS
SELECT * FROM [logdata];

-----

SELECT [Operation name] , COUNT([Operation name]) as [Operation Count]
FROM [logdata]
GROUP BY [Operation name]
ORDER BY [Operation Count]

/* Common errors

1. External table 'logdata' is not accessible because the location does not exist o
r it is used by another process. Here your Shared Access Signature is an issue.

2. Msg 16544, Level 16, State 3, Line 34
The maximum reject threshold is reached. This happens when you try to select the ro
ws of data from the table. This can happen if the rows are not matching the schema
defined for the table
*/

/* By default, tables are defined as clustered columnstore index.
After a load completes, some of the data rows might not be compressed into the colu
mnstore. To optimize query performance and columnstore compression after a load, re
build the table to force the columnstore index to compress all the rows.
*/
ALTER INDEX ALL ON [EventHistory_Lake] REBUILD;

-- verify the data was loaded into the 60 distributions
-- Find data skew for a distributed table
DBCC PDW_SHOWSPACEUSED('EventHistory');

```

CTAS is a more customizable version of the SELECT...INTO statement. SELECT...INTO doesn't allow you to change neither the distribution method nor the index type as part of the operation. You create the new table by

using the default distribution type of ROUND_ROBIN, and the default table structure of CLUSTERED COLUMNSTORE INDEX.

```
-- Lab - (Dedicated SQL Pool) - External Tables - Parquet

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd@123';

-- Here we are using the Storage account key for authorization

CREATE DATABASE SCOPED CREDENTIAL AzureStorageCredential
WITH
    IDENTITY = 'appdatalake7000',
    SECRET = 'VqJnhLUibasTfhSuAxkgIgy97GjRzHL9VN0PkjD8y+KYzl1LSDCflF6LXLrezAYKL3Mf1bu
LdZoJXa/38BXYA==';

-- In the SQL pool, we can use Hadoop drivers to mention the source

CREATE EXTERNAL DATA SOURCE log_data
WITH (
    LOCATION = 'abfss://data@storageneil.dfs.core.windows.net',
    CREDENTIAL = AzureStorageCredential,
    TYPE = HADOOP
)

-- Here we are mentioning the file format as Parquet

CREATE EXTERNAL FILE FORMAT parquetfile
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);

-- Notice that the column names don't contain spaces
-- When Azure Data Factory was used to generate these files, the column names could
not have spaces

CREATE EXTERNAL TABLE [logdata]
(
    [Id] [int] NULL,
    [Correlationid] [varchar](200) NULL,
    [Operationname] [varchar](200) NULL,
    [Status] [varchar](100) NULL,
    [Eventcategory] [varchar](100) NULL,
    [Level] [varchar](100) NULL,
    [Time] [datetime] NULL,
    [Subscription] [varchar](200) NULL,
    [Eventinitiatedby] [varchar](1000) NULL,
    [Resourcetype] [varchar](1000) NULL,
    [Resourcegroup] [varchar](1000) NULL
)

WITH (
    LOCATION = '/parquet/*.parquet',--select all the parquet files from the folder
```

```

DATA_SOURCE = log_data,
FILE_FORMAT = parquetfile
)

/*
A common error can come when trying to select the data, here you can get various errors such as MalformedInput

You need to ensure the column names map correctly and the data types are correct as per the parquet file definition (Data types are embedded)
*/

SELECT * FROM logdata

SELECT [Operation name] , COUNT([Operation name]) as [Operation Count]
FROM logdata
GROUP BY [Operation name]
ORDER BY [Operation Count]

```

2) Loading Data into Dedicated SQL Pool

1) Using the Copy statement

→ Using T-SQL, we can transfer data into a table in a SQL Pool

A mistake that many people make when first exploring dedicated SQL Pools are to use the service administrator account as the one used for loading data. Instead, it's better to create specific accounts assigned to different resource classes dependent on the anticipated task. This will optimize load performance and maintain concurrency as required by managing the available resource slots available within the dedicated SQL Pool.

```

-- Lab - Loading data into a table - COPY Command - CSV

-- Never use the admin account for load operations (keep it only for monitoring and admin purposes)
-- Create a separate user for load operations

-- This has to be run in the master database as we are adding a login and user
CREATE LOGIN user_load WITH PASSWORD = 'Azure@123';

--Here, we are adding a user associated with the login
CREATE USER user_load FOR LOGIN user_load;
GRANT ADMINISTER DATABASE BULK OPERATIONS TO user_load;
GRANT CREATE TABLE TO user_load;
GRANT ALTER ON SCHEMA:::dbo TO user_load;

```

```

/* We can create multiple workload groups in order to provision compute resources such that two different tasks such as a user loading data doesn't use the full resource capacity when a user is present to perform some analysis job*/

CREATE WORKLOAD GROUP DataLoads --Workload Isolation
WITH (
    MIN_PERCENTAGE_RESOURCE = 80
    ,CAP_PERCENTAGE_RESOURCE = 100
    ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 4 -- factor of 80 (guaranteed more than 20 concurrencies)
);
--[Max Concurrency] = [CAP_PERCENTAGE_RESOURCE] / [REQUEST_MIN_RESOURCE_GRANT_PERCENT]

CREATE WORKLOAD CLASSIFIER [ELTLogin] --Workload Classification
WITH (
    WORKLOAD_GROUP = 'DataLoads'
    ,MEMBERNAME = 'user_load'
    ,IMPORTANCE = High -- Workload Importance
);

-- Drop the external table if it exists
DROP EXTERNAL TABLE logdata

-- Create a normal table
-- Login as the new user and create the table
-- Here I have added more constraints when it comes to the width of the data type

CREATE TABLE [logdata]
(
    [Id] [int],
    [Correlationid] [varchar](200) ,
    [Operationname] [varchar](200) ,
    [Status] [varchar](100) ,
    [Eventcategory] [varchar](100) ,
    [Level] [varchar](100) ,
    [Time] [datetime] ,
    [Subscription] [varchar](200) ,
    [Eventinitiatedby] [varchar](1000) ,
    [Resourcegroup] [varchar](1000) ,
    [Resourcegroup] [varchar](1000)
)

```

2) Azure Synapse Pipeline

Define pipelines to carry out copying activity

1. Go to the **Synapse Studio** (web.azuresynapse.net) and click on data
2. Click on the + icon to connect to an external data source
3. Multiple options like Blob storage, Cosmos DB, Data Lake Gen2 etc are provided. Select the appropriate one, here is Gen2

4. It will open a new linked service page where all the details have to be filled in and then click on create
5. In order to fetch the data from the external table, we need to provide some additional access to services which was not required when viewing the data as admin in the Gen2 storage
6. Go to Access Control inside the Gen2 space, and add a role assignment with **Blob Data Contributor**
7. Coming back to **Synapse Studio**, we will be able to see both dedicated SQL pool data as well as external data
8. Now we can view all data and run SQL queries for any activity.
9. Since we want to load data, we can right-click on any file where we want to append/ copy the data and click on New SQL script → Bulk Load
10. Fill in the required configuration settings and continuing forward, we will have a SQL query auto-generated that can be used for copying the data

3) Using Polybase to define external tables

Here data from an external table can be copied into internal tables

PolyBase requires the following elements:

1. An external data source that points to the `abfss` path in ADLS Gen2 where the Parquet files are located
2. An external file format for Parquet files
3. An external table that defines the schema for the files, as well as the location, data source, and file format

```
-- Lab - Loading data using PolyBase

CREATE LOGIN user_load WITH PASSWORD = 'Azure@123';

CREATE USER user_load FOR LOGIN user_load;
GRANT ADMINISTER DATABASE BULK OPERATIONS TO user_load;
GRANT CREATE TABLE TO user_load;
GRANT ALTER ON SCHEMA::dbo TO user_load;

CREATE WORKLOAD GROUP DataLoads
WITH (
    MIN_PERCENTAGE_RESOURCE = 100
```

```

    ,CAP_PERCENTAGE_RESOURCE = 100
    ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 100
  );

CREATE WORKLOAD CLASSIFIER [ELTLogin]
WITH (
    WORKLOAD_GROUP = 'DataLoads'
    ,MEMBERNAME = 'user_load'
);

-- Here we are following the same process of creating an external table

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'P@ssw0rd@123' ;

-- If you want to see existing database scoped credentials
SELECT * FROM sys.database_scoped_credentials

CREATE DATABASE SCOPED CREDENTIAL AzureStorageCredential
WITH
    IDENTITY = 'appdatalake7000',
    SECRET = 'VqJnhlUibasTfhSuAxxgIgy976jRzHL9VN0PkjD8y+KYzl1LSDCf1F6LXlrezAYKL3Mf1buL
dZoJXa/38BXYA==';

-- If you want to see the external data sources
SELECT * FROM sys.external_data_sources

--Step 1
CREATE EXTERNAL DATA SOURCE log_data
WITH (
    LOCATION = 'abfss://data@appdatalake7000.dfs.core.windows.net',
    CREDENTIAL = AzureStorageCredential,
    TYPE = HADOOP
)

-- If you want to see the external file formats
SELECT * FROM sys.external_file_formats

--Step 2
CREATE EXTERNAL FILE FORMAT parquetfile
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);

-- Create the external table as the admin user
--Step 3
CREATE EXTERNAL TABLE [logdata_external]
(
    [Id] [int] NULL,
    [Correlationid] [varchar](200) NULL,
    [Operationname] [varchar](200) NULL,

```

```

[Status] [varchar](100) NULL,
[Eventcategory] [varchar](100) NULL,
[Level] [varchar](100) NULL,
[Time] [datetime] NULL,
[Subscription] [varchar](200) NULL,
[Eventinitiatedby] [varchar](1000) NULL,
[Resourcetype] [varchar](1000) NULL,
[Resourcegroup] [varchar](1000) NULL
)
WITH (
    LOCATION = '/parquet/',
    DATA_SOURCE = log_data,
    FILE_FORMAT = parquetfile
)

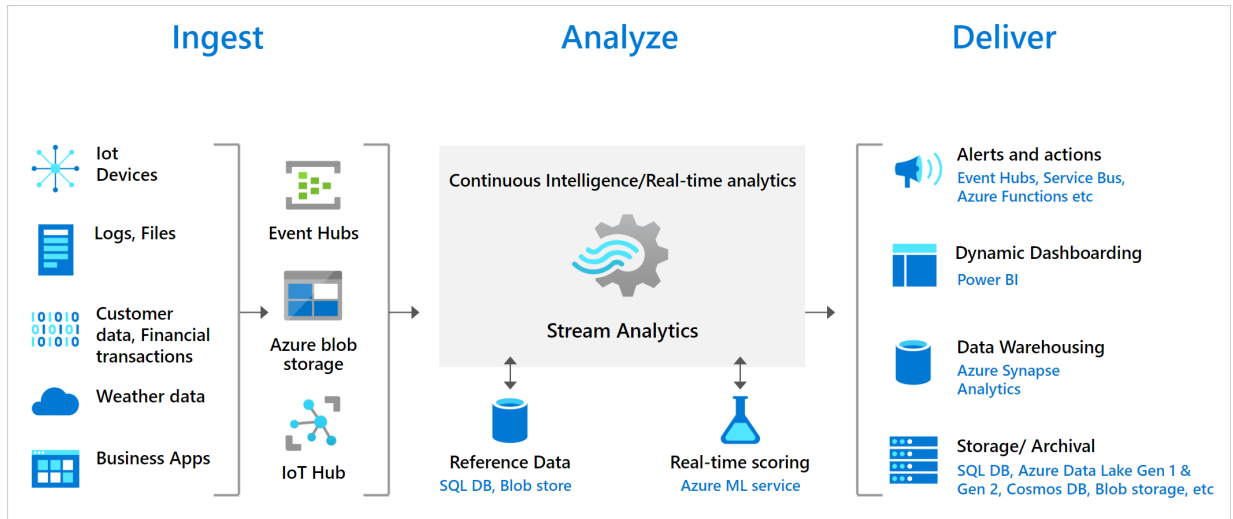
-- Now create a normal table by selecting all of the data from the external table

CREATE TABLE [logdata]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED INDEX (id)
)
AS
SELECT *
FROM [logdata_external];

```

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.

What are data streams

Data streams:
In the context of analytics, data streams are event data generated by sensors or other sources that can be analyzed by another technology

Data stream processing approach:
There are two approaches. Reference data is streaming data that can be collected over time and persisted in storage as static data. In contrast, streaming data have relatively low storage requirements. And run computations in sliding windows

Data streams are used to:

Analyze data:
Continuously analyze data to detect issues and understand or respond to them

Understand systems:
Understand component or system behavior under various conditions to fuel further enhancements of said system

Trigger actions:
Trigger specific actions when certain thresholds are identified

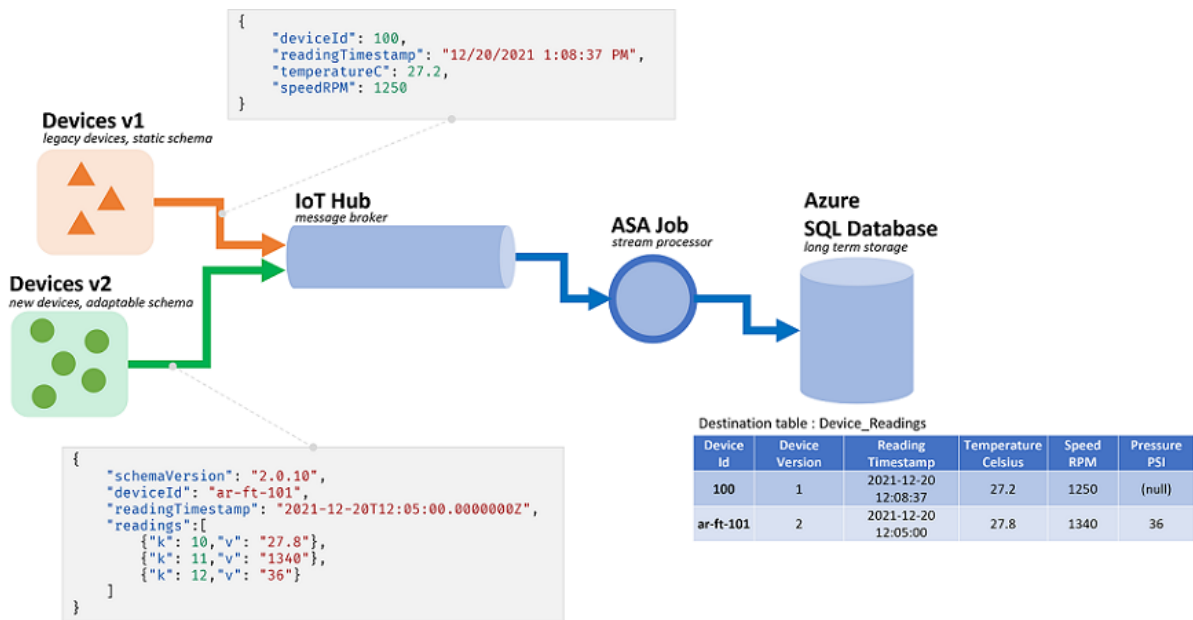
✓ Create a Stream Analytics job ([link](#))

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  
)  
  
SELECT * 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](#)

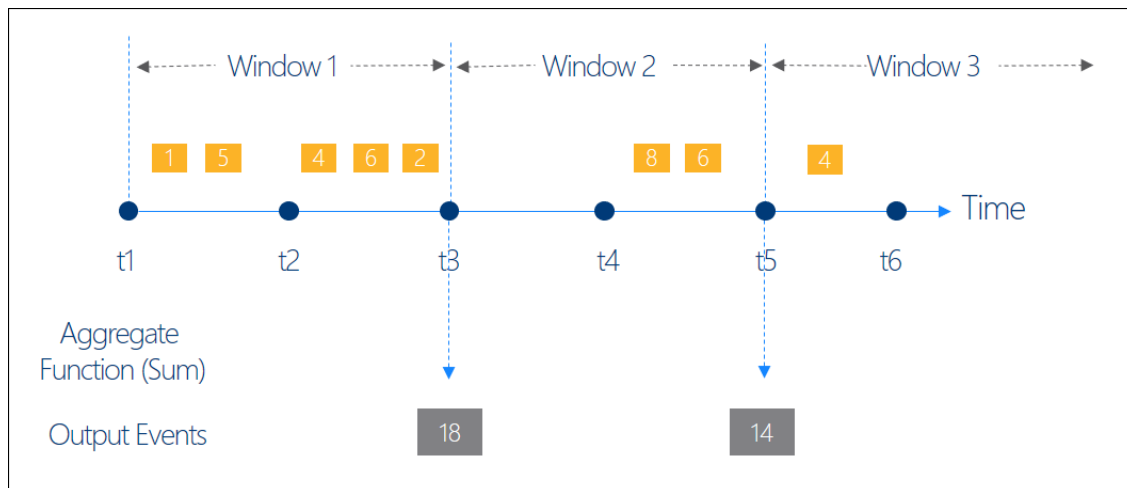
3. **Temporal analytic functions:** `ISFIRST`, `TOPONE`, `LAST`, and `LAG` with `LIMIT DURATION`

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 seconds

A 10-second Tumbling Window

`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

```
--Example of a query using tumbling windows

/*The query averages the engine temperature and speed over a two-second duration by adding TumblingWindow(Duration(second, 2)) to the query's GROUP BY clause. Then it selects all telemetry data, including the average values from the previous step, and specifies the anomalies as new fields

The query outputs all fields from the anomalies step into the powerBIAAlerts output where aggressivedriving = 1 or enginetempanomaly = 1 or oilanomaly = 1 for reporting. The query also aggregates the average engine temperature and speed of all vehicles over the past two minutes, using TumblingWindow(Duration(minute, 2)), and outputs these fields to the synapse output.*/

WITH Averages AS (
    SELECT
        AVG(engineTemperature) averageEngineTemperature,
        AVG(speed) averageSpeed
    FROM
        eventhub TIMESTAMP BY [timestamp]
    GROUP BY
        TumblingWindow(Duration(second, 2))
),
Anomalies AS (
    select
        t.vin,
        t.[timestamp],
        t.engineTemperature,
        a.averageEngineTemperature,
        a.averageSpeed,
        t.engineoil,
        t.accelerator_pedal_position,
        t.brake_pedal_status,
        t.transmission_gear_position,
        (CASE WHEN a.averageEngineTemperature >= 405 OR a.averageEngineTemperature <= 1
5 THEN 1 ELSE 0 END) AS enginetempanomaly,
        (CASE WHEN t.engineoil <= 1 THEN 1 ELSE 0 END) AS oilanomaly,
        (CASE WHEN (t.transmission_gear_position = 'first' OR
            t.transmission_gear_position = 'second' OR
            t.transmission_gear_position = 'third') AND
            t.brake_pedal_status = 1 AND
            t.accelerator_pedal_position >= 90 AND
            a.averageSpeed >= 55 THEN 1 ELSE 0 END) AS aggressivedriving
    FROM eventhub t TIMESTAMP BY [timestamp]
    INNER JOIN Averages a ON DATEDIFF(second, t, a) BETWEEN 0 And 2
),
VehicleAverages AS (
```

```

SELECT
    AVG(engineTemperature) averageEngineTemperature,
    AVG(speed) averageSpeed,
    System.TimeStamp() AS snapshot
FROM
    eventhub TIMESTAMP BY [timestamp]
GROUP BY
    TumblingWindow(Duration(minute, 2))
)

-- INSERT INTO POWER BI
SELECT
    *
INTO
    powerBIA Alerts
FROM
    Anomalies
WHERE aggressivedriving = 1 OR enginemetpanomaly = 1 OR oilanomaly = 1

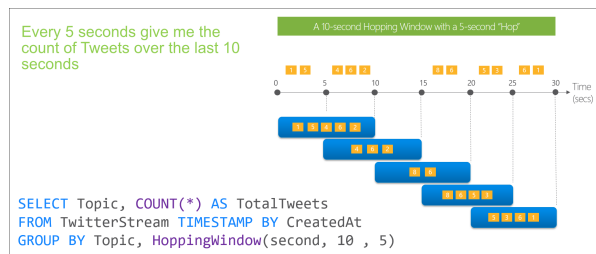
-- INSERT INTO SYNAPSE ANALYTICS
SELECT
    *
INTO
    synapse
FROM
    VehicleAverages

```

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.

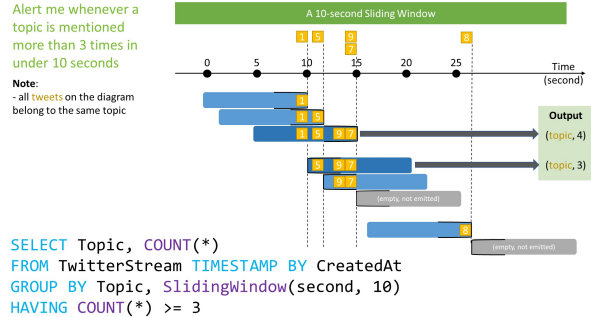
The `windowSize` is 10 seconds, and the `hopSize` is 5 seconds



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

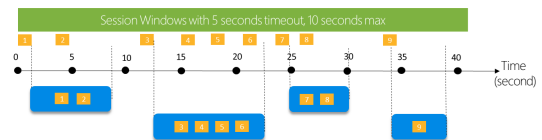
words, when an event enters or exits the window. **So, every window has at least one event.**



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



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`

```
-- Output the count of events that occur within 2 minutes of each other with a maximum
duration of 60 minutes.
```

```

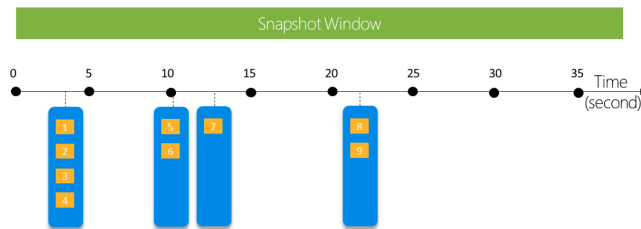
SELECT
  Username,
  MIN(ClickTime) AS WindowStart,
  System.Timestamp() AS WindowEnd,
  DATEDIFF(s, MIN(ClickTime), System.Timestamp()) AS DurationInSeconds
FROM Clickstream TIMESTAMP BY ClickTime
GROUP BY Username, SessionWindow(minute, 2, 60) OVER (PARTITION BY Username)

```

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

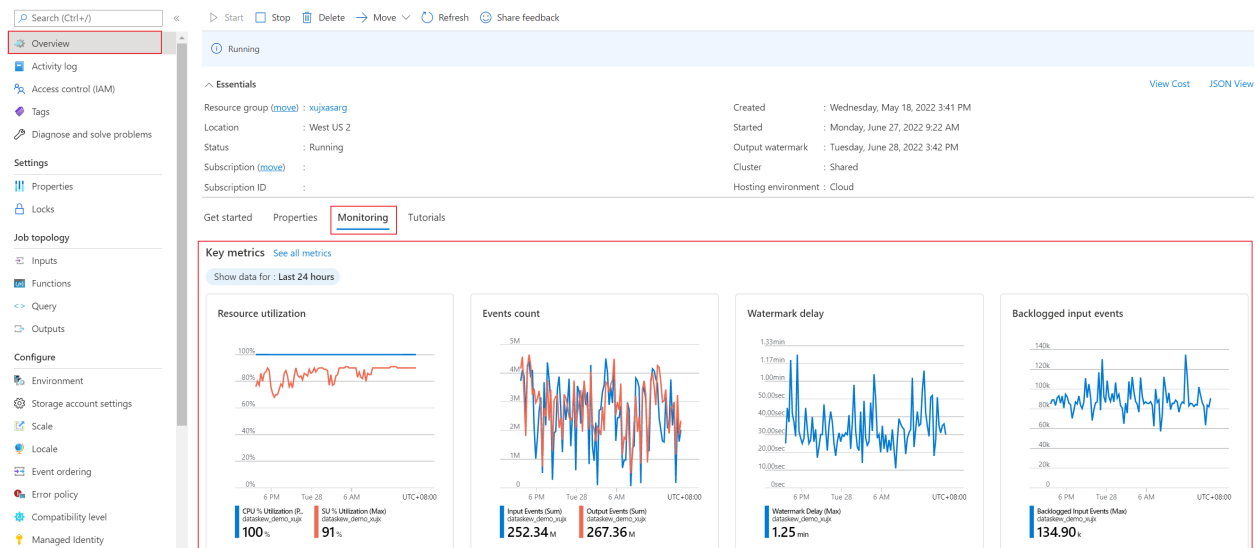


```

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

```

Monitoring Performance + Metrics



Some of the important metrics:

SU% Utilization

Percentage of memory that your job utilizes. If this metric is consistently over 80 percent, the watermark delay is rising, and the number of backlogged events is rising, consider increasing streaming units (SUs) and/or scale with query parallelization.

Runtime Error

The total number of errors related to query processing. Examine the activity or resource logs and make appropriate changes to the inputs, query, or outputs.

Watermark delay (Important)

This metric is aimed towards providing a reliable signal of job health which is agnostic to input and output patterns of the job.

Modern stream processing systems differentiate between event time also referred to as application time, and arrival time. **EVENT TIME** is the time generated by the producer of the event and typically contained in the event data as one of the columns. **ARRIVAL TIME** is the time when the event was received by the event ingestion layer, for example, when the event reaches Event Hubs.

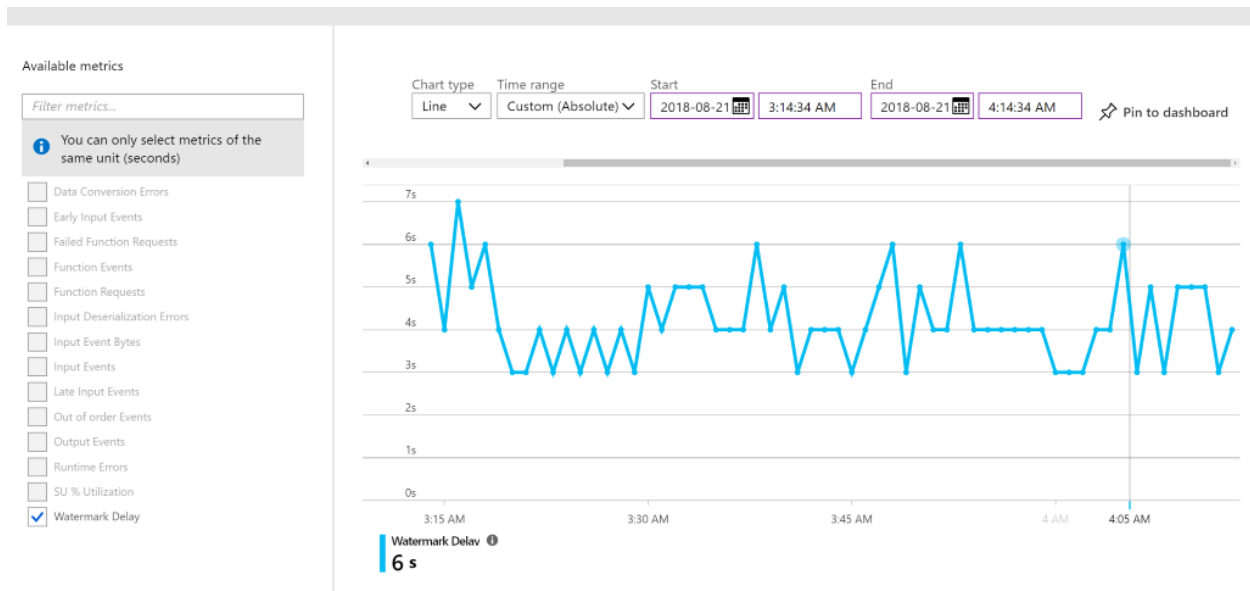
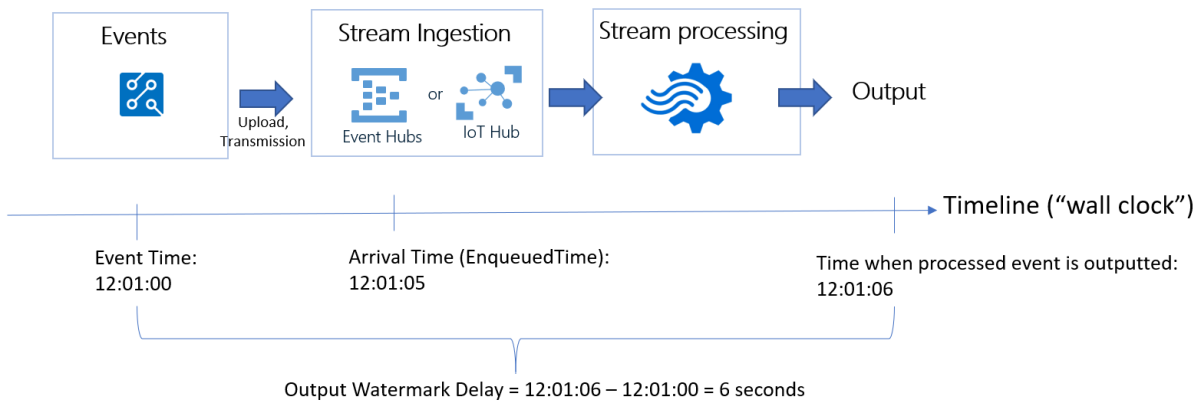
Most applications prefer to use event time as it excludes possible delays associated with transferring and processing of events. In-Stream Analytics, you can use the **TIMESTAMP BY** clause to specify what value should be used as event time.

For example, when Stream Analytics reports a certain watermark value at the output, it guarantees that all events prior to this timestamp were already computed. Watermark can be used as an indicator of liveness for the data produced by the job. If the delay between the current time and the watermark is small, it means the job is keeping up with the incoming data and produces results defined by the query on time.

Below we show an illustration of this concept using a simple example of a passthrough query:

Simple case: no time window, late arrival and out-of-order policy set to 10 seconds

```
SELECT *  
FROM input TIMESTAMP BY eventTime
```



This value represents the maximum watermark delay across all partitions of all outputs in the job.

Input deserialization error

The number of input events that couldn't be deserialized. Examine the activity or resource logs and make appropriate changes to the input

Backlogged Input events

The number of input events that are backlogged. A nonzero value for this metric implies that your job can't keep up with the number of incoming events. If this value is slowly increasing or is consistently nonzero, you should scale out your job.

✓ Azure Event Hub

Azure Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters. It can also be configured to scale dynamically, when required, to handle increased throughput.

Event Hubs is one of three types of message brokers available on Azure. Message brokers act as intermediaries between event producers, such as mobile phone apps, and event consumers, like dashboards or data processing pipelines.

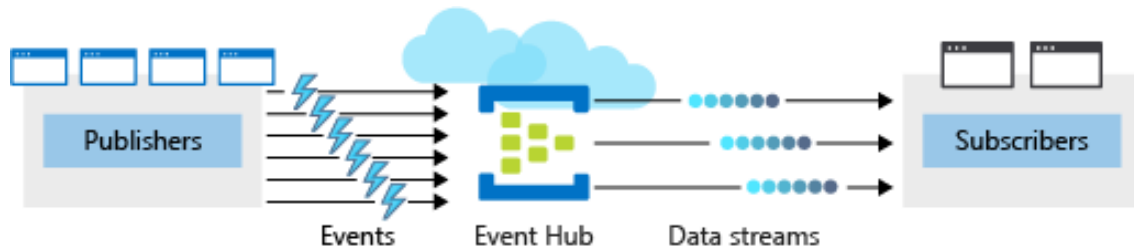
Event processing

The process of consuming data streams, analyzing them, and deriving actionable insights out of them is called Event Processing and has three distinct components:

Event producer	Examples include sensors or processes that generate data continuously such as a heart rate monitor or a highway toll lane sensor
Event processor	An engine to consume event data streams and deriving insights from them. Depending on the problem space, event processors either process one incoming event at a time (such as a heart rate monitor) or process multiple events at a time (such as a highway toll lane sensor)
Event consumer	An application which consumes the data and takes specific action based on the insights. Examples of event consumers include alert generation, dashboards, or even sending data to another event processing engine

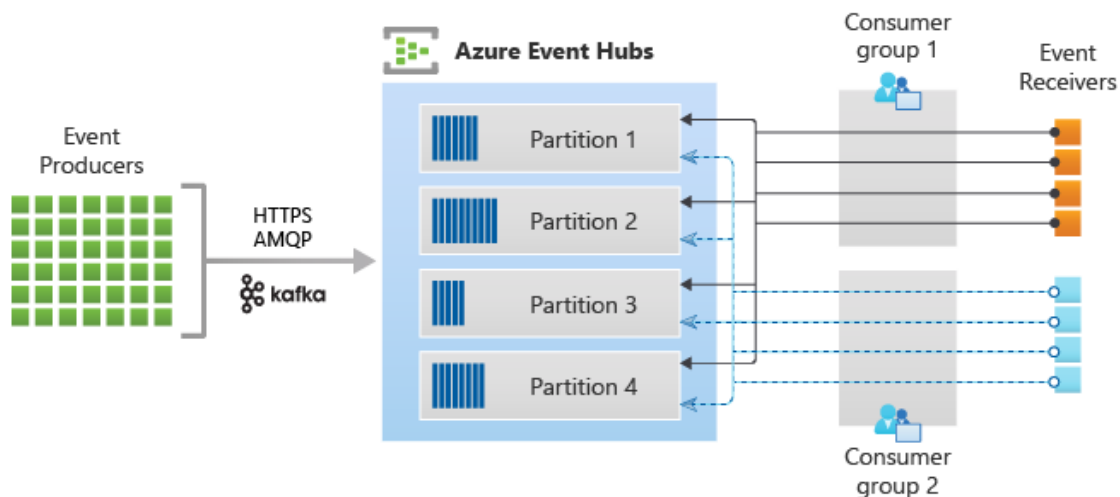
Live Data Processing should be able to **ingest** high volumes of data, **process** these data using sufficient processing power, and **generate** output data in real-time that will get stored in storage with high bandwidth

- An entity that sends data to your event hub is called a ***publisher or producer***
- An entity that reads data from an event hub is called a ***consumer***, or a ***subscriber***. Each ***consumer group*** can independently seek and read data, from each partition, at their own pace.
- An ***event*** is a small packet of information (a *datagram*) that contains a notification. Events can be published individually or in batches, but a ***single publication can't exceed 1 MB***.



Event publishers are any app or device that can send out events using either HTTPS, Advanced Message Queuing Protocol (AMQP) 1.0, or Apache Kafka.

- For publishers that send data frequently, **AMQP** has better performance. ([More reading](#))
- For more intermittent publishing, **HTTPS** is the better option.



Temporal Decoupling

The temporal decoupling provided by message brokers means that the event producer and event consumers don't need to run concurrently.

Load Balancing

Event Hubs is able to handle sudden influxes of traffic than a directly coupled consumer that needs to spend time processing each message. As consumers pull data at their own rate, they avoid being overloaded at any given moment and can process any backlog during moments of lower traffic

Partition

A **partition** is an ordered sequence of events that are held in an Event Hub Partitions can be used to divide or prioritize work and ensure that certain types of data are physically stored together for ease of processing and backup.

Auto-Inflate automatically scales the number of Throughput Units assigned to your Standard Tier Event Hubs Namespace when your traffic exceeds the capacity of the Throughput Units assigned to it. You can specify a limit to which the Namespace will automatically scale.

Checkpointing

It is a process by which readers mark or commit their position within a partition event sequence. Checkpointing is the responsibility of the consumer and occurs on a per-partition basis within a consumer group.



Pull Model

Event Hubs guarantees message caching, but the responsibility for reading that cache falls to the consumer application. This makes it the responsibility of the consumer(s) to ensure data are processed before they expire. This provides flexibility but also can mean that messages are lost in exceptional circumstances.

Create and configure an event hub

There are two main steps to creating a new event hub.

1. The first step is to **define** the Event Hubs **namespace**. An Event Hubs namespace is a container for managing one or more event hubs.
2. The second step is to **create an event hub** in that namespace. The following parameters are required to create an event hub:
 - **Event hub name** - Event hub name that is unique within your subscription
 - **Partition count** - The number of partitions required in an event hub (between 2 and 32 for the standard tier). The partition count should be directly related to the expected number of concurrent consumers and **can't be changed** after the hub has been created. **If not defined, the value defaults to 4.**
 - **Message retention** - The number of days (1 to 7 for the standard tier) that messages will remain available if the data stream needs to be replayed for any reason. **If not defined, this value defaults to 7.** For Event Hubs **Premium** and **Dedicated**, the maximum retention period is **90 days**.

Create Event Hub ...

Event Hubs

Capture Details

Azure Event Hubs Capture enables you to automatically deliver the streaming data in Event Hubs to an Azure Blob storage or Azure Data Lake Store account of your choice, with the added flexibility of specifying a time or size interval. Setting up Capture is fast, there are no administrative costs to run it, and it scales automatically with Event Hubs throughput units. Event Hubs Capture is the easiest way to load streaming data into Azure, and enables you to focus on data processing rather than on data capture. [Learn more about capture.](#)

Capture

On Off

i Note: Enabling Capture will result in additional charges to this account. [Learn more about our pricing.](#)

Time window (minutes)

5

Size window (MB)

300

Do not emit empty files when no events occur during the Capture time window

Capture Provider

Azure Storage Account

Azure Storage Container *

[Select Container](#)

Storage Account

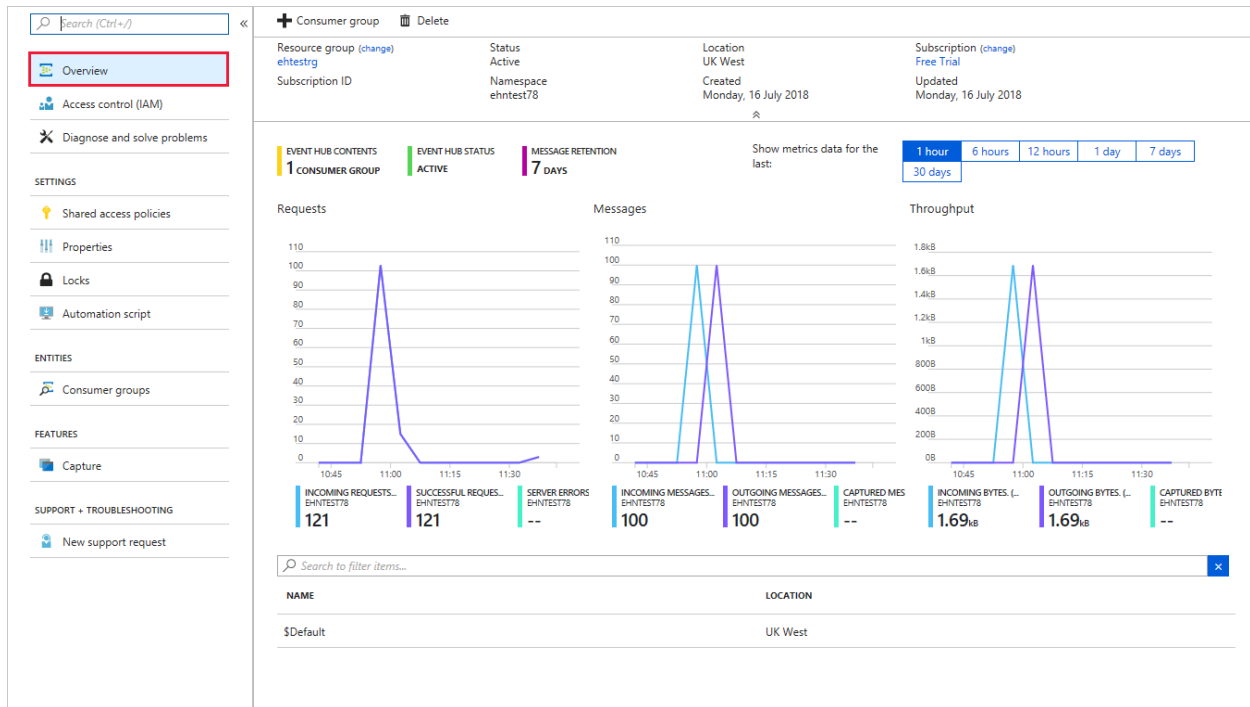
Sample Capture file name formats

{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{H...

Capture file name format ⓘ

{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}...

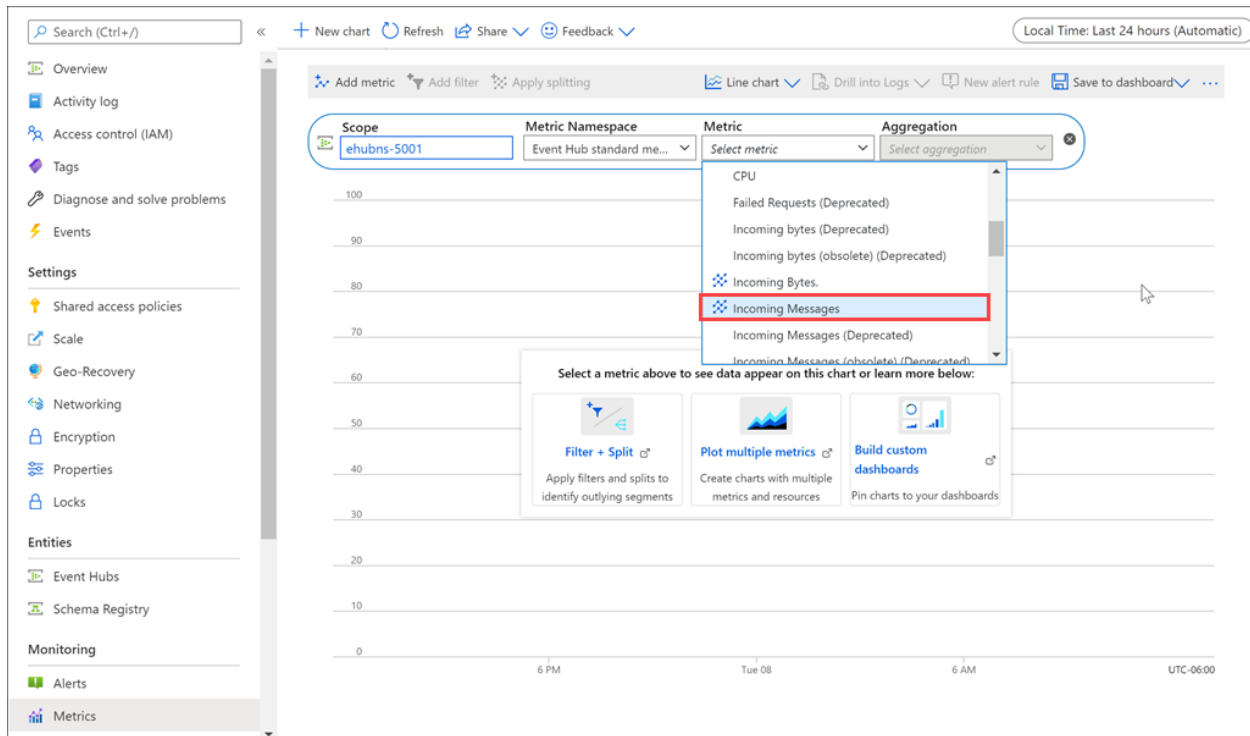
Monitor Performance



The Overview pane for your Event Hub service shows message counts, which represent the data (events) received and sent by the event hub.

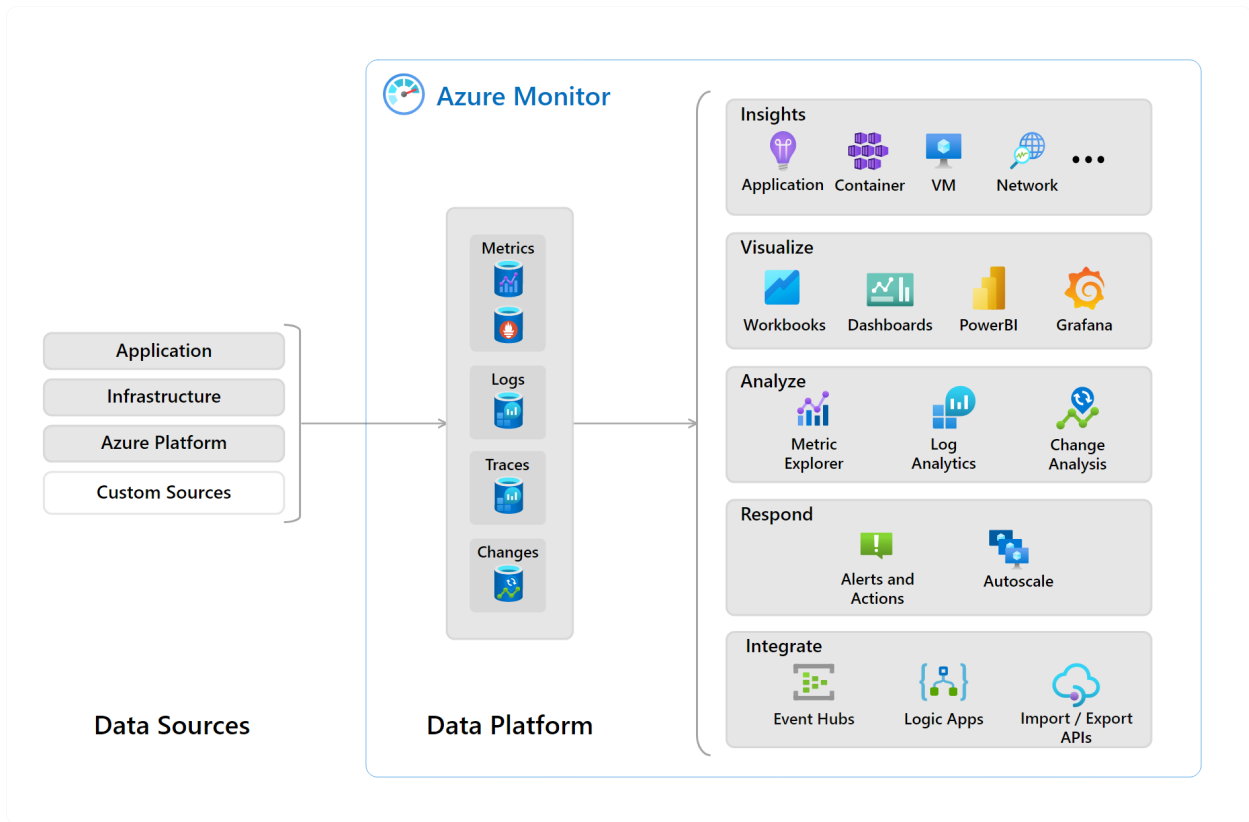
Useful **metrics** available in Event Hubs include:

- **Throttled Requests:** The number of throttled requests because the throughput exceeded unit usage.
- **ActiveConnections:** The number of active connections on a namespace or Event Hub.
- **Incoming/Outgoing Bytes:** The number of bytes sent to/received from the Event Hubs service over a specified period.



✓ Azure Monitor

Centralized management and consolidated monitoring of all azure resources. Azure Monitor groups together other services like Metrics, Alerts, Activity Log, etc



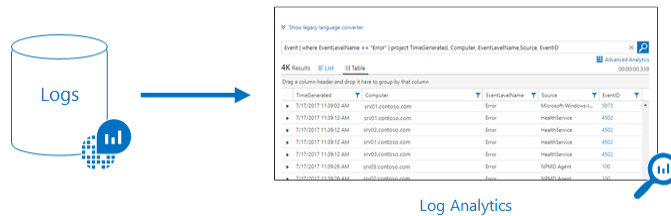
Metrics

Metrics are numerical values that describe some aspect of a system at a particular point in time.



Logs

Logs are events that occurred within the system. They can contain different kinds of data and may be structured or free-form text with a timestamp.



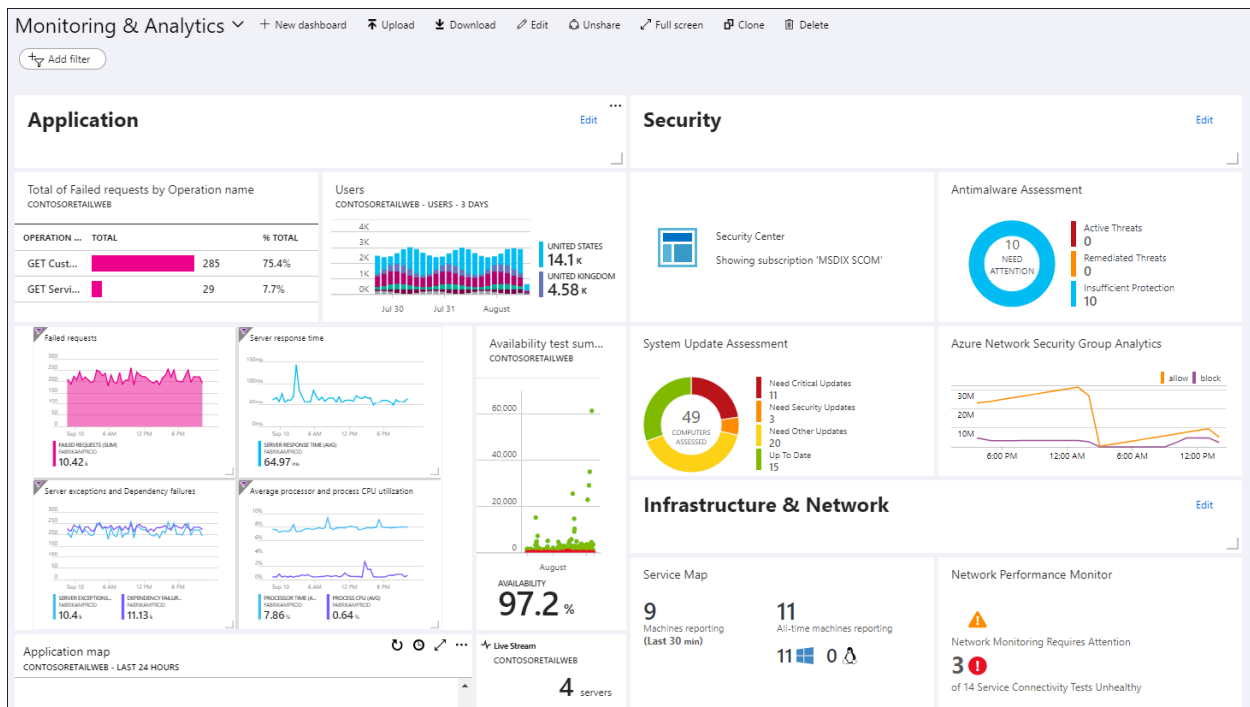
Traces

Traces are a series of related events that follow a user request. They can be used to determine the behavior of application code and the performance of different transactions.

While logs will often be created by individual components of a distributed system, a trace measures the operation and performance of your application across the entire set of components.

Changes

Changes are a series of events that occur in your Azure application and resources.



Azure Dashboards allows combining different kinds of data