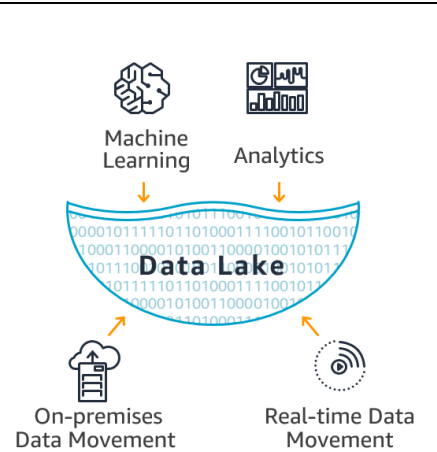


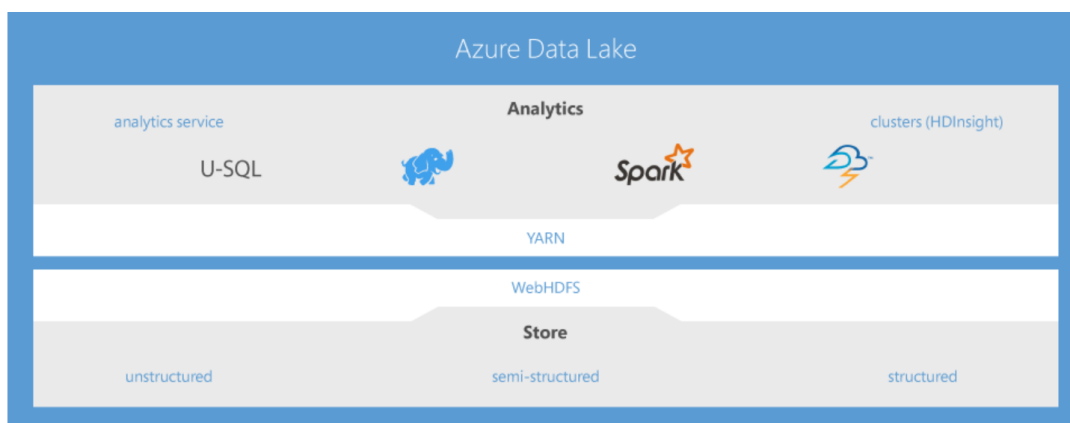
Data Lake Storage

- Data lake storage is **storage solution** for Big Data
- **Centralized repository** for all enterprise data
- Data Stored in **natural** format.
- Built on **HDFS** Standard



Azure Data Lake

- Azure Data Lake is **highly scalable** data storage and analytical cloud service provided by Microsoft.
- Azure data lake forms **core** of all big data services that Microsoft offer within Azure.



- It can store and analyse **petabyte-size files and trillions of objects**.

- It not only provides massive storage but operate with high performance. Storage is optimized for analytical workloads.
- Azure Data Lake includes all the capabilities required to make it easy for **developers, data scientists and analysts** to store data of any size, shape, and speed, and do all types of processing and analytics across platforms and languages.
- Data can be accessed in various ways (programming, SQL-like queries, and REST calls).
- It provides **enterprise grade security**.
- Data Lake is a **cost-effective** solution to run big data workloads.
- It has three primary services
 1. Azure Data Lake Store
 2. Azure Data Lake Analytics
 3. Azure HDInsight

Azure Data Lake Storage

- Enterprise wide **hyperscale repository** for big data analytic workloads that require large throughput
- Azure Data Lake Store is **built on open Hadoop File System (HDFS)** Standard, supports WebHDFS.
- Store data of any size, type, or ingestion speed in one single place for operational and exploratory analytics.
- **Unlimited storage** without restricting file sizes or data volumes.
- To process data up to Petabytes in size, ADLS distributes data across multiple nodes where mappers and reducers, in the magnitude of thousands, process this data in parallel to provide fast results.

Note: The Protocol WebHDFS, is based on an industry-standard RESTful mechanism that does not require Java binding. It works with operations such as reading files, writing to files, making directories, changing permissions, and renaming. It defines a public HTTP REST API, which permits clients to access HDFS over the Web

Cloud storage in Azure Data Lake implementation have two options

1. Data lake storage Gen1(will be retired on **Feb 29, 2024**)
2. Azure storage (Specifically blob storage)

Data Lake Storage Gen1 features:

- File system semantics
- File-level granular security
- Scale
- Optimized for analytic workload

Azure Storage Features:

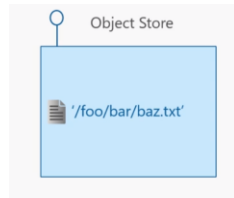
- Geo-redundancy
- High availability and disaster recovery
- Hot/cold/archive tiers.

Azure Data Lake Storage Gen2:

- Built on Azure Blob Storage.
- **Enable hierarchical namespace** (HNS) property for general purpose V2, storage account

Object Storage (Azure Blob Storage):

Data objects are stored in flat namespace



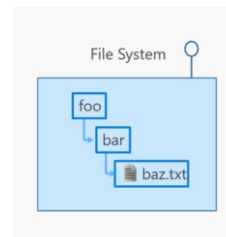
Object storage is known for being highly scalable and very cost effective.

Azure data lake storage Gen 1:

Hierarchical file system

Data can be organized within directory structure

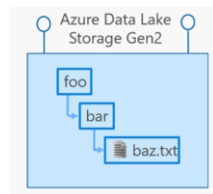
Security implemented at directory and file level files in folders.



This is optimized for analytics workloads.

Azure data lake storage Gen 2(multi modal storage service):

Features from Azure Data Lake Storage Gen1, such as file system semantics, directory, and file level security, capabilities for analytics workloads and scale are combined with low-cost, tiered storage (Hot/Cool/Archive), high availability/disaster recovery (RA-GRS) capabilities from Azure Blob storage.





Refer:

<https://channel9.msdn.com/Shows/Learn-Azure/Understanding-Azure-Data-Lake-Storage-Gen-2/player?format=ny>

Lab1: Create Data Lake Storage Gen 2

1. Portal→All Services→Storage→Storage Accounts

Basics Tab

Basics Advanced Networking Data protection Tags Review + create

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *

Resource group * [Create new](#)

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ *

Region ⓘ *

Performance ⓘ *

☒ Standard: Recommended for most scenarios (general-purpose v2 account)

☐ Premium: Recommended for scenarios that require low latency.

Redundancy ⓘ *

→Next:Advanced

2. Advanced Tab

Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace ☒

3. Keep other default settings and Create

Lab2: On-prem to Data Lake Gen2 Using Portal and Storage Explorer

Upload data Using Portal


1. Data Storage→Containers→+Container(New Container)→
Name: testdata→Create
2. Go to container→Upload→Browse Files→Select some file→Upload

Upload blob

×

testdata/

Files ⓘ



☐ Overwrite if files already exist

▼ Advanced

Upload Data Using Storage Explorer

1. Launch Azure Storage Explorer from Local machine →Connect to Storage
2. Select storage→Select testdata container→Drag and drop multiple files
from windows to container

Data Lake Gen 2 Features:

1. Hadoop compatible access:

- You can treat the data as if it's stored in a Hadoop Distributed File System. With this feature, you can **store the data in one place and access it through compute technologies** including Azure Databricks, Azure HDInsight, and Azure Synapse Analytics without moving the data between environments.
- The Azure Blob File System (ABFS) driver provides the interface to ADLS Gen2 storage.
- The **ABFS driver (used to access data)** is available within all Apache Hadoop environments. These environments include Azure HDInsight, Azure Databricks, and Azure Synapse Analytics.
- The ABFS driver, included in the Databricks Runtime, supports standard file system semantics on Azure Blob storage.

2. Security:

- The security model for Data Lake Gen2 supports **ACL and POSIX** permissions along with some extra granularity specific to Data Lake Storage Gen2.
- You can set **permissions** at a **directory level** or **file level** for the data stored within the data lake.
- Settings may be configured through Storage Explorer or through frameworks like Hive and Spark.
- All **data** that is stored is **encrypted** at rest by using either Microsoft or customer-managed keys.

3. Hierarchical Namespace:

- Hierarchical namespaces organize blob data into *directories* and stores metadata about each directory and the files within it. This

structure allows operations, such as directory renames and deletes, to be performed in a single atomic operation.

- Hierarchical namespaces keep the data organized, which yields better storage and retrieval performance for an analytical use case.
- It also provides easier navigation and as a result Data processing requires less computational resources, reducing both the time and cost.

4. Cost Effective:

- Data Lake Storage Gen2 offers low-cost storage capacity and transactions. Using **built in feature lifecycle**, the cost can be kept minimum.

Note: As Azure Data Lake Storage Gen2 is integrated into the Azure Storage platform, applications can use either the Blob APIs or the Azure Data Lake Storage Gen2 file system APIs to access data.

URI scheme to reference data

1. ADLS Gen2 Connectivity – File System

`abfs[s]://file_system@account_name.dfs.core.windows.net/<dir1>/<dir2>/<file_name>`

2. Azure Blob storage connectivity

`wsab[s]://containername@account_name.dfs.core.windows.net/<folder1>/<folder2>`

Note:

Windows Azure Storage Blob driver([WASB](#)) provided the original support for Azure Blob Storage.

This driver performed the complex task of mapping file system semantics (as required by the Hadoop FileSystem interface) to that of the object store style interface exposed by Azure Blob Storage.

The ABFS driver was designed to overcome the inherent deficiencies of WASB.

Refer: <https://docs.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-abfs-driver>

Using the URI

Using the above URI format, standard Hadoop tools and frameworks can be used to reference these resources:

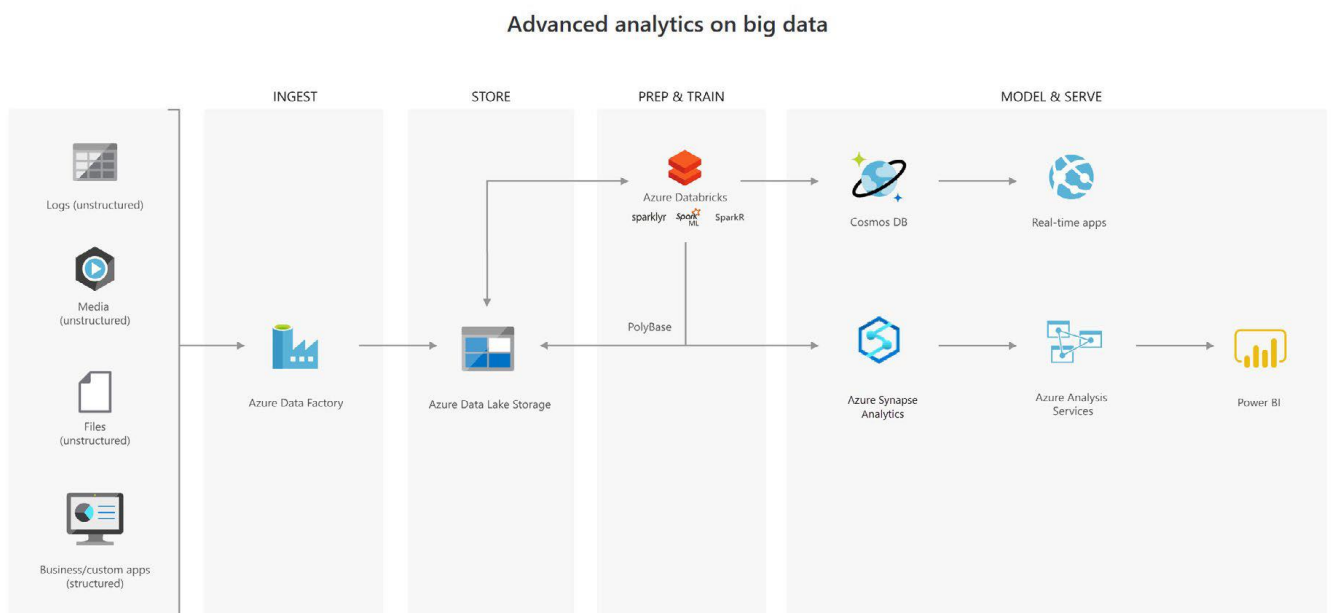
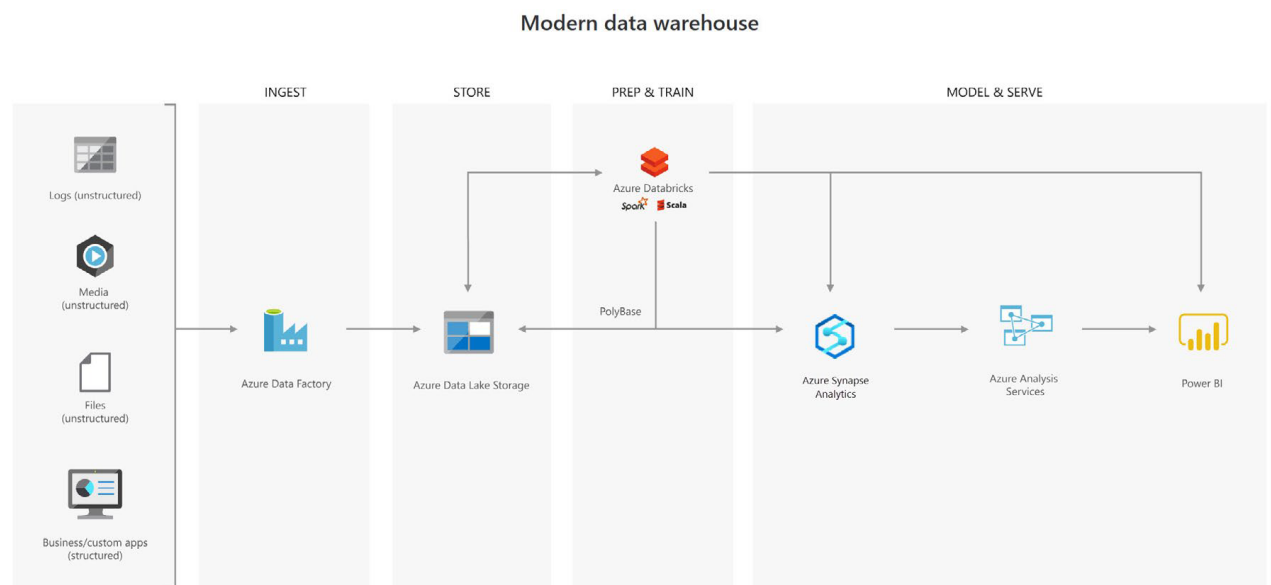
```
hdfs dfs -mkdir -p
abfs://fileanalysis@myanalytics.dfs.core.windows.net/tutorials/flightdelays/
data
hdfs dfs -put flight_delays.csv
abfs://fileanalysis@myanalytics.dfs.core.windows.net/tutorials/flightdelays/
data/
```

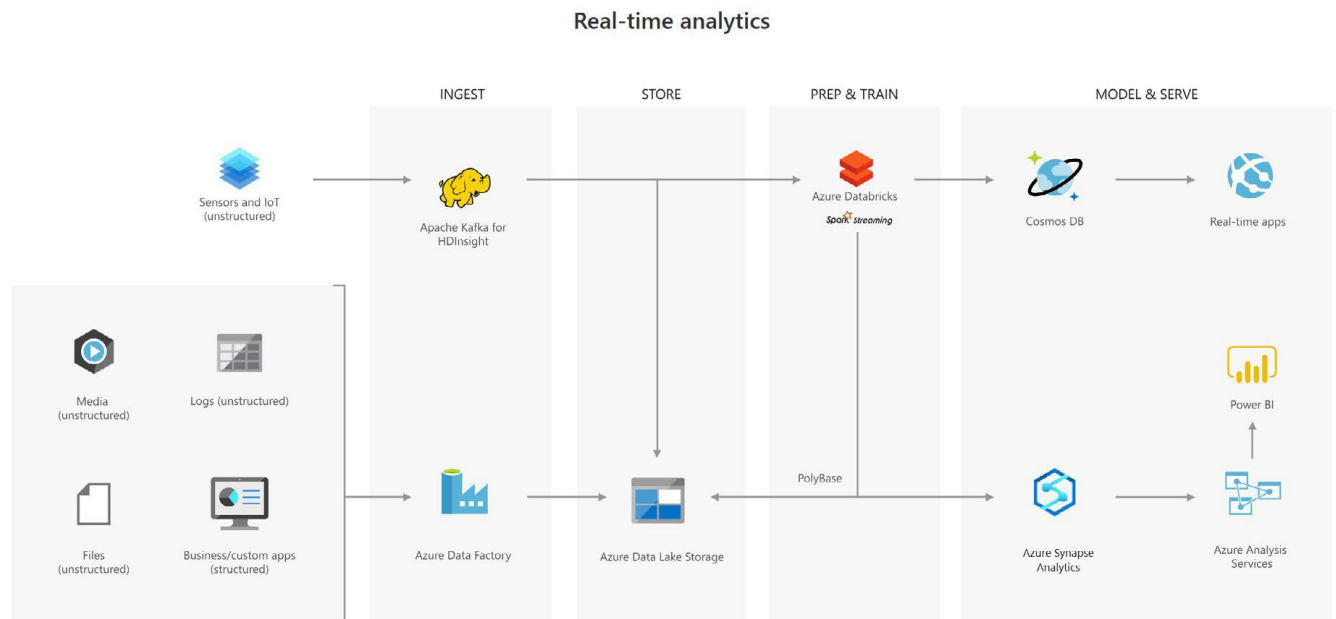
Consideration for Hierarchical Namespace (Enabled/Disabled):

- If you want to store data *without performing analysis on the data*, set the **Hierarchical Namespace** option to **Disabled** to set up the storage account as an Azure Blob storage account. You can also use blob storage to archive rarely used data or to store website assets such as images and media.

- If you are performing analytics on the data, set up the storage account as an Azure Data Lake Storage Gen2 account by setting the **Hierarchical Namespace** option to **Enabled**

Azure Data Lake Store Use Cases





Data Lake Gen2 Security

Azure Data Lake Storage Gen2 implements an access control model that supports both **Azure role-based access control (RBAC)** and **POSIX-like access control lists (ACLs)**.

Role-based access control:

- RBAC uses role assignments to effectively apply sets of permissions to *security principals*.
- Security principal can be **user, group, service principal, or managed identity** that is defined in Azure Active Directory (AD) that is requesting access to Azure resources.
- Using RBAC role assignments is a powerful mechanism to control access permissions, it is a very coarsely grained mechanism relative to ACLs. The

smallest granularity for RBAC is at the container level and this will be evaluated at a higher priority than ACLs

- If the same user/application is granted both RBAC and ACL permissions, the RBAC role (for example Storage Blob Data Contributor which allows you to read, write and delete data) will override the access control list rules and no additional ACL checks are performed.
- If the security principal does not have an RBAC assignment, or the request's operation does not match the assigned permission, then ACL checks are performed to determine if the security principal is authorized to perform the requested operation.

Access control lists on files and directories:

- Access control lists provide access to data at the folder or file level and allows for a far more fine-grained data security system
- They enable POSIX style security, which means that permissions are stored on the items themselves. Each file and directory in your storage account has an access control list.
- You can associate a security principal with an access level for files and directories. These associations are captured in an *access control list (ACL)*.

Levels of Permission

- Read: Read contents of file.
- Write: Write contents of file.

- Execute: It is in context with directory, required for child items in directory.

Read + execute at directory level is required to list the content of directory

Write + execute directory level is required to create child items in directory

Permissions inheritance:

- Permissions are only inherited if default permissions have been set on the parent items before the child items have been created as it is POSIX style model i.e. permissions are stored on the items themselves
- For ACL, it is a best practice, and recommended, to create a security group in AAD and maintain permissions on the group rather than individual users. Using this structure will allow you to add and remove users or service principals without the need to reapply ACLs to an entire directory structure) Instead, you simply need to add or remove them from the appropriate Azure AD security group

Access control list types:

There are two kinds of access control lists: *access ACLs* and *default ACLs*.

- **Access ACLs** control access to an object. Files and directories both have access ACLs.
- **Default ACLs** are templates of ACLs associated with a directory that determine the access ACLs for any child items that are created under that directory. Files do not have default ACLs.
- Changing the default ACL on a parent does not affect the access ACL or default ACL of child items that already exist.

Note: Access control via ACLs is enabled for a storage account as long as the Hierarchical Namespace (HNS) feature is turned ON.

Refer: <https://docs.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-access-control>

Shared Key and SAS:

- Azure Data Lake Storage Gen2 supports Shared Key and SAS methods for authentication
- With Shared key full access to all operations on all resources is provided. The caller effectively gains 'super-user' access, meaning full access to all operations on all resources, including setting owner and changing ACLs.
- SAS tokens include allowed permissions as part of the token

Blob storage folder structure vs Data Lake Folder Structure

Blob storage	Data Lake
Customer	2020
Cust_2020_01_01.json	01
Cust_2020_01_02.json	Cust_2020_01_01.Parquet
	Cust_2020_01_02.Parquet
Cust_2020_02_01.json	02
	Cust_2020_02_01. Parquet
Cust_2020_03_01.json	03
	Cust_2020_03_01. Parquet
Cust_2020_04_01.json	04

	Cust_2020_04_01. Parquet
--	--------------------------

Advantages of Better Folder Structure

- Easy access
- Better organization
- Helps faster analytical queries from Azure Databricks and Azure Data warehouse
- Better role-based access control and ACLs

Designing Azure Data Lake Storage Structure

General Factors for planning Data Lake

- What kind of data?
- Who will access ?(Security)
- Access pattern

Designing Directory Structure

Folder Structure: Should represent the partition of data by folder.

IoT Workloads:

General Layout:

`*{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/*`

Example:

Telemetry for an airplane engine within the UK

- `*UK/Planes/BA1293/Engine1/2017/08/11/12/*`

You can secure data using (ACL) based on region and subject matter to specific users as data is at the end of hierarchy

Batch Workloads:

- For **Raw data** use “in” Directory
- For **Processed data** use “out” Directory.
- For **Corrupted data** use “bad” Directory
- **Parent-level directories** for things such as **region and subject matters** (for example, organization, product, or producer)

- Date and time in the structure for better organization, filtered searches, security, and automation in the processing.
- Granularity for the date structure is determined by the interval on which the data is uploaded or processed, such as hourly, daily, or even monthly.

Example:

```
{Region}/{SubjectMatter(s)}/In/{yyyy}/{mm}/{dd}/{hh}/*\ *
```

```
{Region}/{SubjectMatter(s)}/Out/{yyyy}/{mm}/{dd}/{hh}/*\ *
```

```
{Region}/{SubjectMatter(s)}/Bad/{yyyy}/{mm}/{dd}/{hh}/*
```

*Note: There is no need of **In or output Folder** when data is being processed directly into databases like Hive or SQL Server as output already goes into a separate folder for the Hive table or external database.*

Time Series:

- For Hive workloads, partition pruning of time-series data can help some queries read only a subset of the data, which improves performance.
- Pipelines that ingest time-series data, often place their files with a structured naming for files and folders.

Example:

```
\DataSet\YYYY\MM\DD\datafile_YYYY_MM_DD.tsv
```

```
\DataSet\YYYY\MM\DD\HH\mm\datafile_YYYY_MM_DD_HH_mm.tsv
```

File Formates Consideration:

Avro, Parquet, ORC

- Optimized for storing and processing structured data.
- machine-readable binary file formats
- Compressed
- Schema embedded

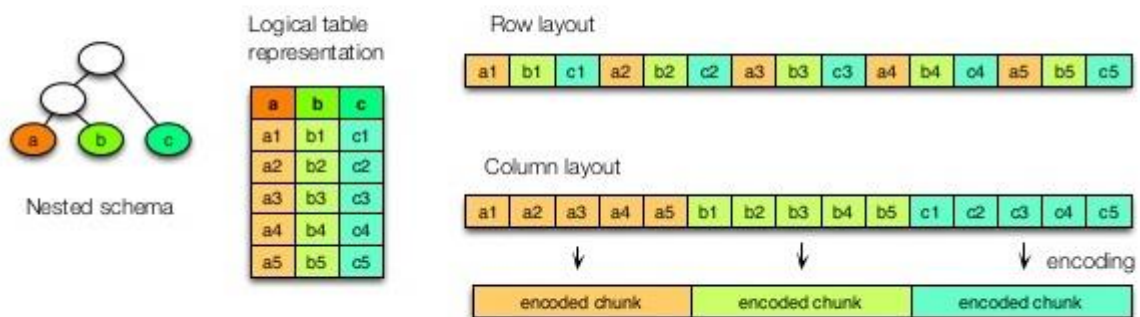
Avro:

- Stores data in a row-based format.
- Consider **Avro** file format in cases where your I/O patterns are more write heavy, or the query patterns favor retrieving multiple rows of records in their entirety.
- Works well with Event Hub or Kafka that write multiple events/messages in succession.

Parquet and ORC

- Columnar storage
- Supports efficient data compression and encoding schemes that can lower data storage costs.
- Services such as [Azure Synapse Analytics](#), [Azure Databricks](#) and [Azure Data Factory](#) have native functionality that take advantage of Parquet file formats.

Columnar storage



Note: Consider Parquet and ORC, when the I/O patterns are more read heavy or when the query patterns are focused on a subset of columns in the records.

Recommended File Types:

Raw Data:

- Data from relational databases :CSV
- Data from web APIs and NoSQL databases :JSON

Refined Versions Of Data:

- For Querying:Parquet

Consideration for File Size:

- Analytics engines such as HDInsight have a per-file overhead that involves tasks such as listing, checking access, and performing various metadata operations.
- Organize your data into larger sized files for better performance (256 MB to 100 GB in size) as data stored as many small files, can negatively affect performance.

- There should be some process(Aggregation pipeline) to aggregate Raw data with lots of small sized files into larger ones for use in downstream applications.
- Consider saving these large files in a read-optimized format such as [Apache Parquet](#) for downstream processing.

Azure Blob Vs Data Lake:

<u>Blob</u>	<u>Data Lake</u>
Flat Namespace i.e. data stored in single Hierarchy.	Hierarchical namespaces organize blob data into <i>directories</i> and stores metadata about each directory and the files within it.
Flat namespaces, by contrast, require several operations proportionate to the number of objects in the structure.	operations, such as directory renames and deletes, to be performed in a single atomic operation.
Data used not for performing analytics. You can also use blob storage to archive rarely used data or to store website assets such as images and media. can be accessed using HTTP or HTTPS.	Data used for performing analytics. integrated into the Azure Storage platform, applications can use either the Blob APIs or the Azure Data Lake Storage Gen2 file system APIs to access data.