Data Lake Playbook

Table of Contents

[Overview 4](#_Toc512959168)

[Data Lake Architecture 4](#_Toc512959169)

[Data sources 4](#_Toc512959170)

[Storage 5](#_Toc512959171)

[Data movement 5](#_Toc512959172)

[Analytics 5](#_Toc512959173)

[Azure Data Lake Store folders 6](#_Toc512959174)

[/RAW schema 6](#_Toc512959175)

[/RAW/SYSTEM ingestion 7](#_Toc512959176)

[/RAW/EXTERNAL ingestion 8](#_Toc512959177)

[/RAW/FILES ingestion 8](#_Toc512959178)

[/PROCESS and /OUTPUT folder schemas 9](#_Toc512959179)

[/INNOVATION folder schemas 9](#_Toc512959180)

[Error handling folders 10](#_Toc512959181)

[Azure Data Lake Store accounts 10](#_Toc512959182)

[ADLS Access Control 11](#_Toc512959183)

[ADLS ACLS 11](#_Toc512959184)

[ADLS folder default ACLS 13](#_Toc512959185)

[ADLS ACLs limits and Azure AD Groups 14](#_Toc512959186)

[ADLS IAM roles 15](#_Toc512959187)

[Azure Key Vault 15](#_Toc512959188)

[Azure Storage Explorer 16](#_Toc512959189)

[Azure Data Factory 17](#_Toc512959190)

[Azure Data Factory artifacts 18](#_Toc512959191)

[Pipelines 19](#_Toc512959192)

[Activity 19](#_Toc512959193)

[Data Set 19](#_Toc512959194)

[Azure Data Factory naming conventions 19](#_Toc512959195)

[Artifact suffixes 20](#_Toc512959196)

[Connector suffixes 20](#_Toc512959197)

[Integration runtime naming 20](#_Toc512959198)

[Linked service naming 21](#_Toc512959199)

[Pipeline naming 21](#_Toc512959200)

[Data set naming 22](#_Toc512959201)

[Linked services and Azure Key Vault 22](#_Toc512959202)

[Azure Key Vault linked service setup 22](#_Toc512959203)

[ADF Key Vault secret selection in Linked Services 26](#_Toc512959204)

[Azure Databricks linked service setup 27](#_Toc512959205)

[Linked Service for existing Azure Databricks Cluster 28](#_Toc512959206)

[Linked Service for new Azure Databricks Cluster 31](#_Toc512959207)

[Azure Data Factory CSV output format 31](#_Toc512959208)

[Timestamps and pipeline execution 32](#_Toc512959209)

[Azure Data Factory IAM roles 32](#_Toc512959210)

[Azure Databricks and ELT 32](#_Toc512959211)

[Notebook organization and file conventions 32](#_Toc512959212)

[Databricks Azure Data Lake Store mount points 33](#_Toc512959213)

[Spark schema inference 33](#_Toc512959214)

[Spark save folders and metadata file cleanup 34](#_Toc512959215)

[Databricks Databases and Tables 34](#_Toc512959216)

[Spark query performance and optimization 34](#_Toc512959217)

[Azure SQL Database 35](#_Toc512959218)

[Azure SQL Data Warehouse 35](#_Toc512959219)

[Azure Data Catalog 36](#_Toc512959220)

[Dev and Prod Environments 36](#_Toc512959221)

[Deployment and Delivery of Solutions 37](#_Toc512959222)

[Azure Subscriptions and Resource Groups 38](#_Toc512959223)

# Overview

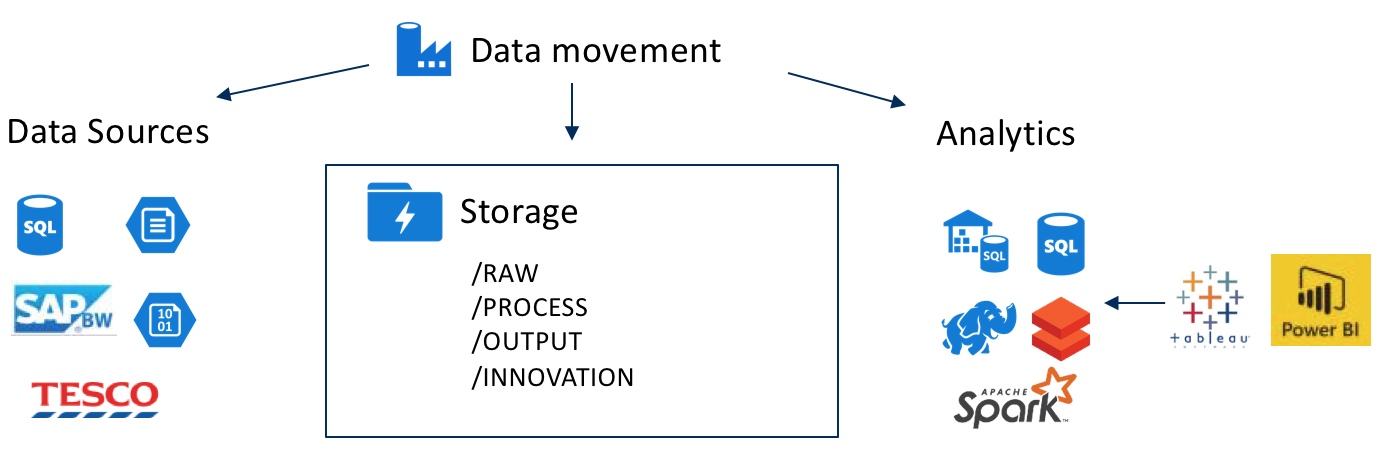
A data lake architecture organizes all of the information of an enterprise in a single location regardless of its format, structure, volume or size. The data lake is a secure source for all reporting and analytical needs and centralizes integration with source systems or organizations. It also provides a solid workspace for data transformation via compute engines and ELT tools so that all datasets have clear data lineage, are cataloged, and discoverable.

Self-service analytics is a key aspiration and desired outcome of building a data lake architecture. Data sources are available for immediate exploitation without the need to wait for updates or changes to percolate through a traditionally slow and difficult change control process inherent in classical Enterprise Data Warehouse ETL designs. Agility is also enabled by giving analytical solutions and qualified individuals the ability to innovate with almost unlimited storage and compute processing capabilities at their fingertips on-demand on the data in the lake without the need for complex development lifecycles.

A successful implementation of a data lake architecture is dependent upon the processes and procedures of an organization. The data lake can easily become a data swamp if good practices are ignored or the wrong set of tools are used without concern for their overall fit. The goal of this document is to provide information, guidance, and procedures for working with the data lake and drive success with analytics projects.

# Data Lake Architecture

The data lake architecture at a high level can be broken down into several areas: data sources, data movement, storage, and analytics platforms.



## Data sources

Data sources includes all systems, processes or personnel that generate data of interest to the business that needs storage in the lake. The sources can range from global systems like SAP BW or SAP Hana, external web services, high velocity input streams, as well as files in file shares that are maintained and updated by . The lake is ready and willing to take on all data shapes size and format to meet the needs for analytics projects.

## Storage

Storage in the data lake rests upon the solid hyper scale cloud platform of Microsoft Azure and its Azure Data Lake Store. The almost unlimited capacity of this system and built in redundancy allow for to store vast quantities of information without worry about disk space or backup tape running out. It also stores all data encrypted at rest and leverages the same Azure AD security model as Office 365 to secure access to any data in its folder structures.

## Data movement

The data lake architecture embraces a modern style of data pipelines for data movement called Extract Load and Transform (ELT). ELT processes large datasets by extracting and loading data from sources and into destinations using scalable movement tools while executing data transformations using code running on scalable distributed compute clusters. The ELT style of data processing can scale from a few small files all the way up to petabyte datasets and beyond while handling a wide variety of data formats and input frequency including real time stream processing.

<picture of data movement>

The primary tools for data movement pipelines in the lake architecture are Azure Data Factory and Azure Databricks. Azure Data Factory orchestrates data movement between source systems and the cloud at scale using a wide variety of connectors including database, file system, web services, and specialized applications systems like SAP. Scheduled execution, orchestration of workflows and complete logging and auditing of all activity is available in ADF to preserve data lineage.

Azure Databricks is the ELT processing tool of choice in the data lake architecture for modifying, cleaning, shaping, and conforming all datasets. Azure Databricks is based on the Spark distributed processing platform with the ability to scale to process massive datasets on large in -memory clusters as well as providing an easy to use development and collaboration environment based on notebooks.

## Analytics

The primary data target from information flowing out of the lake and for consumption via reporting tools is the SQL Database platform in Azure. This Azure database service provides automated management, high availability, backup, data encryption, and row level security and can scale on demand to handle higher workloads and data sizes.

Additional specialized services in the architecture include Azure Databricks for big data analytical processing via Spark and the Azure SQL Data Warehouse system for massively parallel query processing on large datasets.

Azure Data Catalog is used as a metadata repository for describing the format and schema of data flowing into the lake and from the analytics projects. Azure Key Vault is used to store all secrets and connection strings in a secure place that is controlled and audited.

Although a lot of Azure and Microsoft specific tools are recommended, the architecture supports the use of third party ETL/ELT tools that can read and write to Azure Data Lake Store and is completely agnostic to the choice of a reporting tools such as Tableau or Power BI.

# Azure Data Lake Store folders

A clean folder structure that is inherently discoverable is the primary concern in avoiding a data swamp. The folder structure inside all Azure Data Lake Store accounts is managed for purpose, clarity, and function via 4 top level folders.

|  |  |
| --- | --- |
| ***/RAW*** | Ingestion point organized by source system folders under centrally governed data pipelines |
| ***/PROCESS*** | Solution specific subfolders for transformation activity including staging, cleansing or aggregation |
| ***/OUTPUT*** | Solution specific subfolders for final ELT processing output ready for load into a SQL database |
| ***/INNOVATION*** | User specific folders for data science workloads or personal analytics |

Capitalization for all folder and file names in the lake is highly recommended as the lake URLs are case sensitive. Underscores should be used as a replacement character for spaces in names for readability purposes while other special characters are prohibited.

# */RAW* schema

The */RAW* folder area is the destination for all data ingestion and is organized by the source system or entity. The structure of the folders provides information to ease browsing and to clearly segregate details like the specific server, database, table, or folder which sources the data.

|  |  |
| --- | --- |
| ***/RAW/SYSTEM*** | Ingest area for enterprise systems like SAP or JDA organized by system name |
| ***/RAW/EXTERNAL*** | Ingest area for external entities like NIELSEN organized by external name |
| ***/RAW/FILES*** | Ingest areas for internal content organized by solution name |

## /RAW/SYSTEM ingestion

The primary data sources in an enterprise environment are databases that hold operational data created by applications in conjunction with data warehouse environments.

Two URL templates for ADLS folders apply whether extracting an entire table or running a query to extract a portion of the data. The full table option is the recommended option for import as it allows for reuse in all downstream analytics solutions.

These ingestion formats should be used regardless of whether the /RAW files will be creating by connecting directly to the source system or the files are ingested via a storage area like Azure Blob Storage or an internal file share on a periodic basis.

**Full**

/RAW/SYSTEM/{system}/{server}/{database}/{load}/{year}/{month}/{day}/{table}.csv

**Query**

/RAW/SYSTEM/{system}/{server}/{database}/{load}/{solution}/{query}/{year}/{month}/{day}/{query}.csv

|  |  |
| --- | --- |
| system | Name of an application or system |
| server | Server name |
| database | Database/cube/partition of the data source that is on the server |
| load | Full load or delta load of data |
| table | Table name in data source |
| solution | Name of solution when query is used |
| query | Query name describing subset of information being pulled from table |
| year | Year of data import (2018) |
| month | Month number of data import (1-12) |
| day | Day number of data import (1-31) |

Example: Analytics solution Orion Reporting requests information from JDA supply chain databases for EU demand forecasts from server VMWL5166 and database DB1790 for the FCST table:

/RAW/JDA/VMWL5166/DB1790/FULL/2018/4/1/FCST.csv

If the team wanted a subset with a query for the last 180 days for NA a query template would be used to different the request from a more complete data pull and is used for a delta loading process:

/RAW/JDA/VMWL5166/DB1790/DELTA/ORION\_REPORTING/FCST\_NA\_LAST\_180/2018/4/1/ FCST\_NA\_LAST\_180.csv

## /RAW/EXTERNAL ingestion

External entities or systems often need a cloud accessible endpoint to push their content to for storage in the data lake. A secure container in Azure Blob Storage will be created along with a SAS (Shared Access Signature) URL will be provided to them for write access to this location.

/RAW/EXTERNAL/{vendor}/{load}/{year}/{month}/{day}/\*.\*

|  |  |
| --- | --- |
| vendor | Name of vendor |
| load | Full load or delta load of data |
| year | Year of data import (2018) |
| month | Month number of data import (1-12) |
| day | Day number of data import (1-31) |

Example: Nielsen uploads full database extract CSV files to a blob container for ingress into data lake.

/RAW/EXTERNAL/NIELSEN/FULL/2018/04/01/\*.\*

## /RAW/FILES ingestion

A common source of data for solutions will be file shares that contain extracts or content created by employees. They will be copied from their source folder or container on a periodic basis according to the solution that is using them.

/RAW/FILES/{solution}/{share}/{load}/{year}/{month}/{day}/\*.\*

|  |  |
| --- | --- |
| solution | Name of solution |
| share | File share name |
| load | Full load or delta load of data |
| year | Year of data import (2018) |
| month | Month number of data import (01-12) |
| day | Day number of data import (01-31) |

Example: Analytics solution Compass Reporting uploads daily sales summary CSV files in US\_SALES folder on an internal file share created for data lake integration.

/RAW/FILES/COMPASS\_REPORTING/US\_SALES/FULL/2018/04/01/data.csv

# /PROCESS and /OUTPUT folder schemas

The */PROCESS* and */OUTPUT* folders have sub folders representing analytics solutions.

/PROCESS/{solution}

/OUTPUT/{solution}

Example for Compass Reporting solution:

/PROCESS/COMPASS\_REPORTING

/OUTPUT/COMPASS\_REPORTING

Solutions are free to organize these folders according to their specific required in order to clean, join, and maintain their data archives in /PROCESS.

Diagram of movement from /RAW to /PROCESS to /OUTPUT

Solutions are required to maintain a full archive of their content in the /OUTPUT folder and focus their processing on the data files in the lake versus using their relational database as an ETL processing and staging area. A file-based focus on ELT processing is a key part of providing data lineage inside the data lake and allows other projects to reuse the products of a solutions ELT process as a potential input source without the burden and putting load on the reporting relational database.

# /INNOVATION folder schemas

The */INNOVATION* folder will contain subfolders organized by the 5+3 of the employee.

Examples of this folder:

/INNOVATION/MICHADAL

/INNOVATION/TALENTS

The subfolders are under the control of the user and they have flexibility to use them in the manner needed for analytics.

# Error handling folders

Data movement tools in Azure Data Factory have the ability to log offending row errors and continue processing if desired in a pipeline run. This is typically used when ingesting from systems or applications that have prescribed schemas that must be validated during import.

The best practice for this type of operation is to use an ERRORS folder that is located in the root location of the table or object being ingested and a .txt file with the same name as the its table.

An earlier example in the document reference a JDA database import:

/RAW/JDA/VMWL5166/DB1790/FULL/2018/4/1/FCST.csv

Its recommended to put error content in the following location:

/RAW/JDA/VMWL5166/DB1790/FULL/2018/4/1/ERRORS/FCST.txt

# Azure Data Lake Store accounts

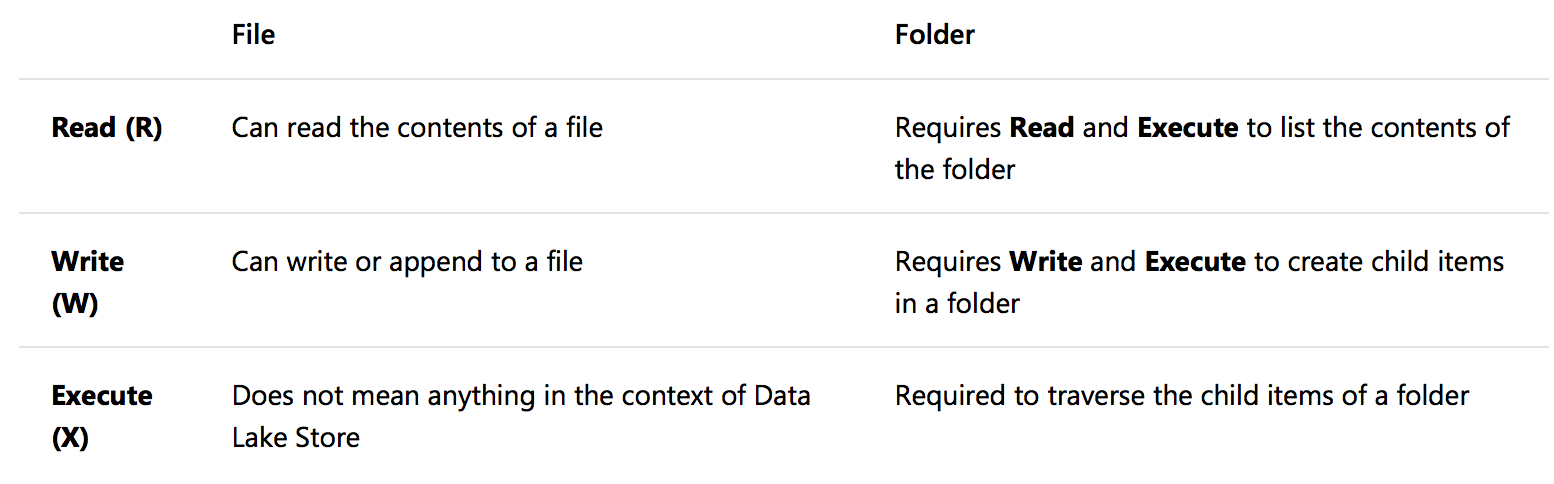
The primary Azure Data Lake Store account and source for all reporting and analytics is Analytics. All analytics projects will work directly from the analytics Azure Data Lake account.

Vault is the only Azure Data Lake Store account used to process sensitive information for masking and anonymizing data. Only after a process of cleaning by centralized secure data movement tools is data transferred from Vault to Analytics. All access to information within Vault and its supported toolset will be strictly controlled.

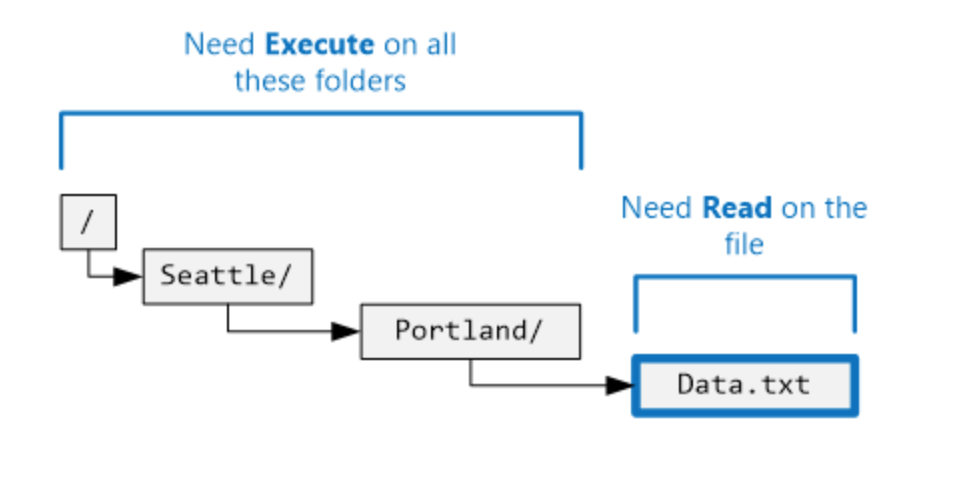
# ADLS Access Control

## ADLS ACLS

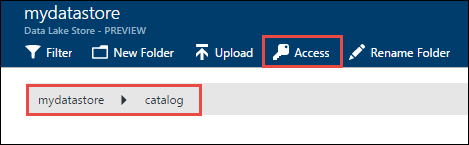
Azure Data Lake Store secures folders and files with permissions assigned to Azure AD users/groups/service principals with POSIX style Access Control Lists. POSIX ACLS contain three primary permissions:



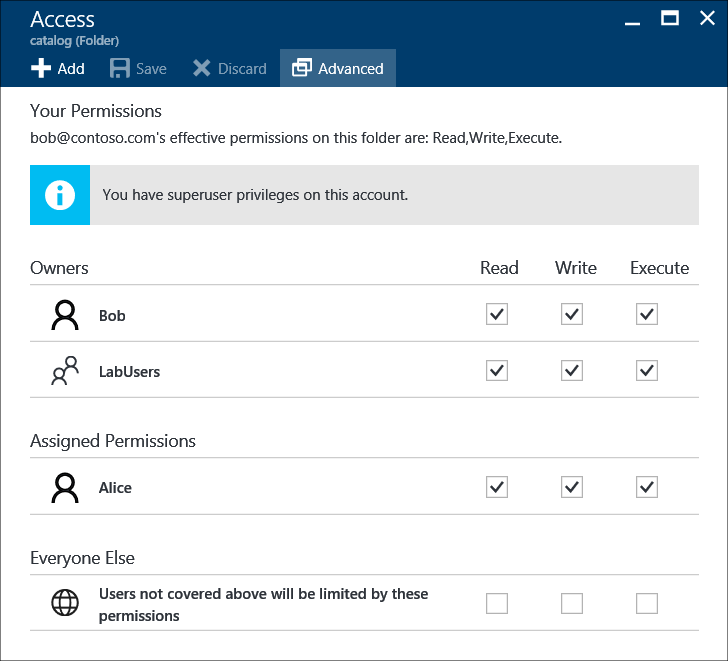
In order to access a lower level file or folder, all folders above the content desired for access must have the appropriate folder Execute permissions enabled on them.



The Data Explorer for an Azure Data Lake Store account in the portal is the best way to set ACLs as there is an Access button at the top for each file or folder.

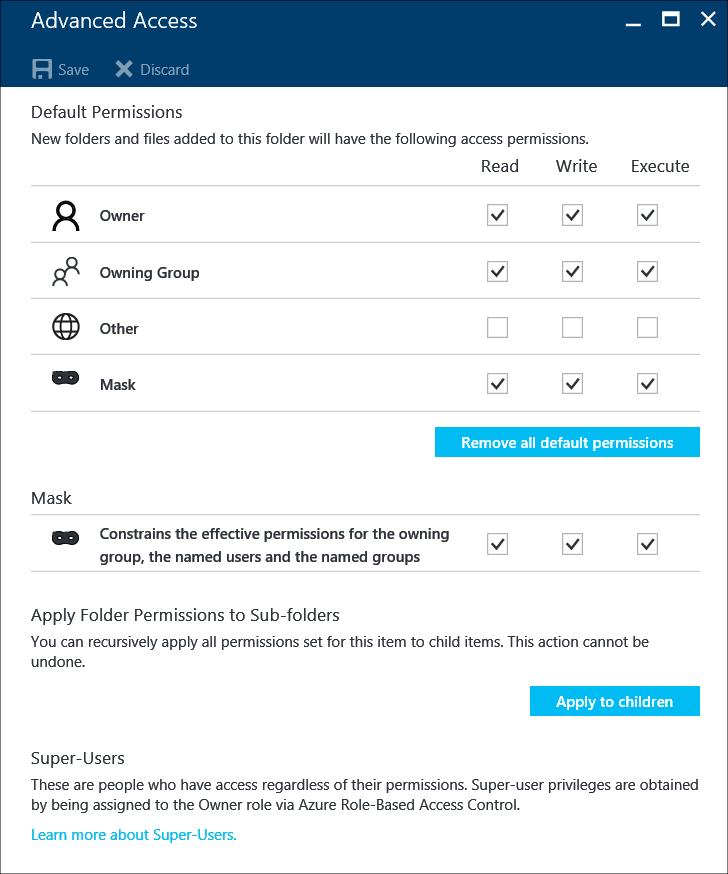


The Access button at top brings up a basic menu for setting access ACL permissions for files and folders.



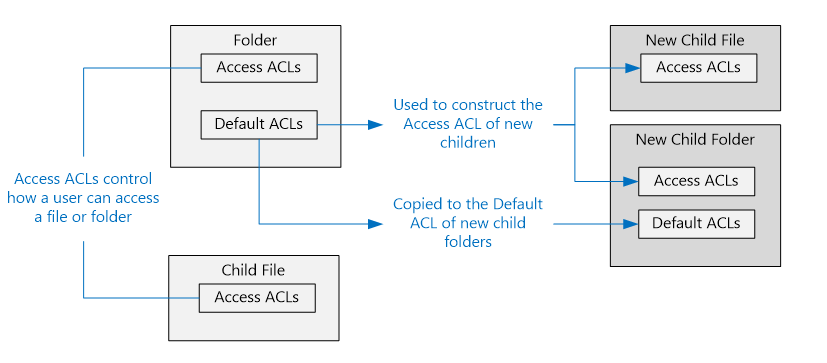
ACLS do not inherit permissions in Azure Data Lake Store. This is different than Windows file permissions inheritance that some users may be accustomed to and requires a bit more administrator of ACLs entries.

The Advanced button at the top of the Access blade for a folder allows you to set entries recursively for all children and to set the folder “default” ACLs when new content is created.



## ADLS folder default ACLS

Folders have an additional “default” ACL list with the same permission structure (Read,Write,Execute) that is used when you create new content like folders and file in an existing directory.



In order to set the default entries for a folder always use the Advanced Access tab.

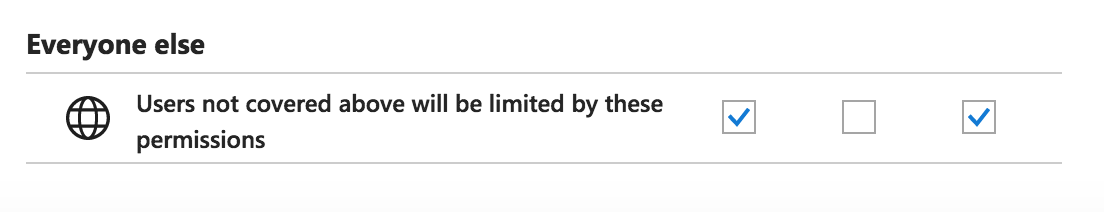
When debugging access control problems with an analytics solution this is one of the first places to check that the defaults were set correctly so that tools like Azure Data Factory doesn’t create unreadable files or folders during data movement.

## ADLS ACLs limits and Azure AD Groups

The ACLs in ADLS files and folders manage access to two primary security entities in Azure: AD users with direct lake access via tools like Azure Storage Explorer, and Azure AD service principals that represent services like Azure Data Factory or Azure Databricks accessing the lake on behalf of other users.

Azure AD Groups are required to manage a large number of Azure AD user or service principal entries that require access as there is a limit of 32 ACL entries per file or folder. This will require careful maintenance as there are no nested groups in Azure AD (what about replicated groups from on prem)???.

All users will be granted read and execute (traverse) privileges for top level folders /RAW, /PROCESS, /OUTPUT, and /INNOVATION to aid discoverability and ease ACL maintenance and limits. At the bottom of the Access blade you can see the default entries for Everyone else.



The /RAW area data sources will have an Azure AD Group for read access and an Azure Group for read/write access in Azure AD to manage who has access to specific source systems. All Azure users and service principals who need access to the folders of that system will have to be added to the group.

The template for /RAW system Azure AD Groups will look like the following:

AZ-{source system}-READ

AZ-{source system}-WRITE

For the JDA system VML5166 database DB1790 the template is filled out with the following:

AZ-JDA-VMWL5166-DB1790-READ

AZ-JDA-VMWL5166-DB1790-WRITE

The subfolders under the analytics solution folders */PROCESS/{solution}* and */OUTPUT/ {solution}* are under full read/write control of for the solution Azure AD group that is created for the solution.

The Azure AD Group template for a solution looks like:

AZ-{solution name}

For the Compass Reporting solution the Azure AD group will be named like:

AZ-COMPASS-REPORTING

## ADLS IAM roles

Full Azure (Identity and Access Management) IAM permissions for Azure Data Lake Store management functions in the Azure console will be limited to infrastructure and AOH personnel. This level of access control is different than the filesystem ACLs used to manage content access inside the Azure Data Lake Store account folders and.

The only Azure IAM roles granted to Azure AD users of the Data Lake will be the Reader role to enable them to use tools like Azure Data Explorer in combination with the access checks that occur via ACLs when content is accessed in the lake.

This Azure Reader role on the account will only be granted to Azure AD groups that are created for the analytics projects or for individuals with personal folders in the /INNOVATION area of the lake who are not in analytics projects.

# Azure Key Vault

Azure Key Vault is a secure storage area for cryptographic keys are storage of application secrets. Azure Data Lake Store accounts are created with the encryption at rest turned on and using encryption keys managed by Azure Key Vault.



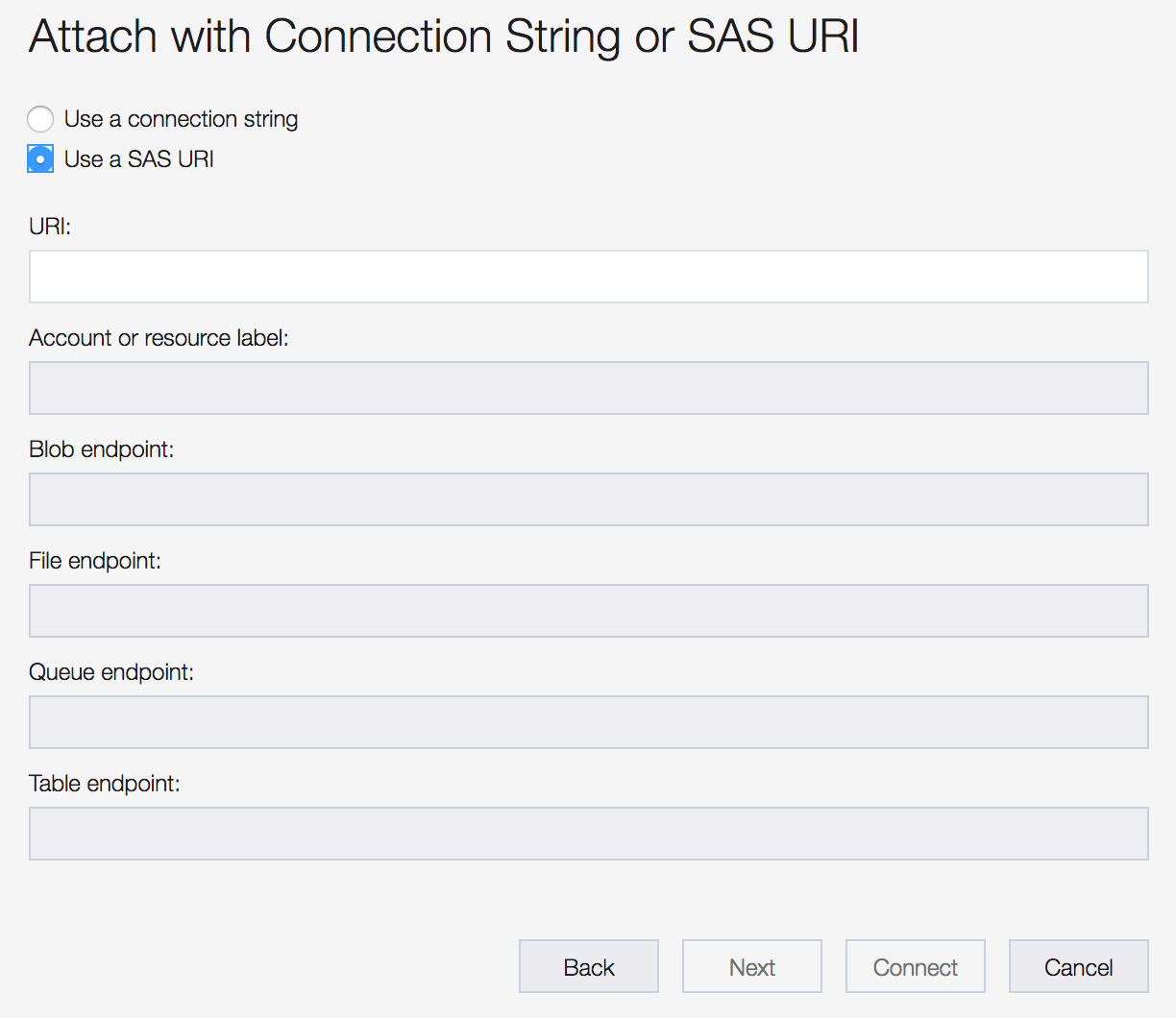
Key Vault is also the repository of all secrets used by Azure Data Factory for its Linked Service data sources and for applications to use with database connection strings to SQL Database or other database systems.

# Azure Storage Explorer

The primary client application for end users to directly access the lake is Azure Storage Explorer. This can be downloaded for free and installed on all platforms and easily configured by logging into a account in the same manner as Office 365.

This tool allows browsing of all folders and files and manipulation for renames and deletes and copies as well as download and upload.

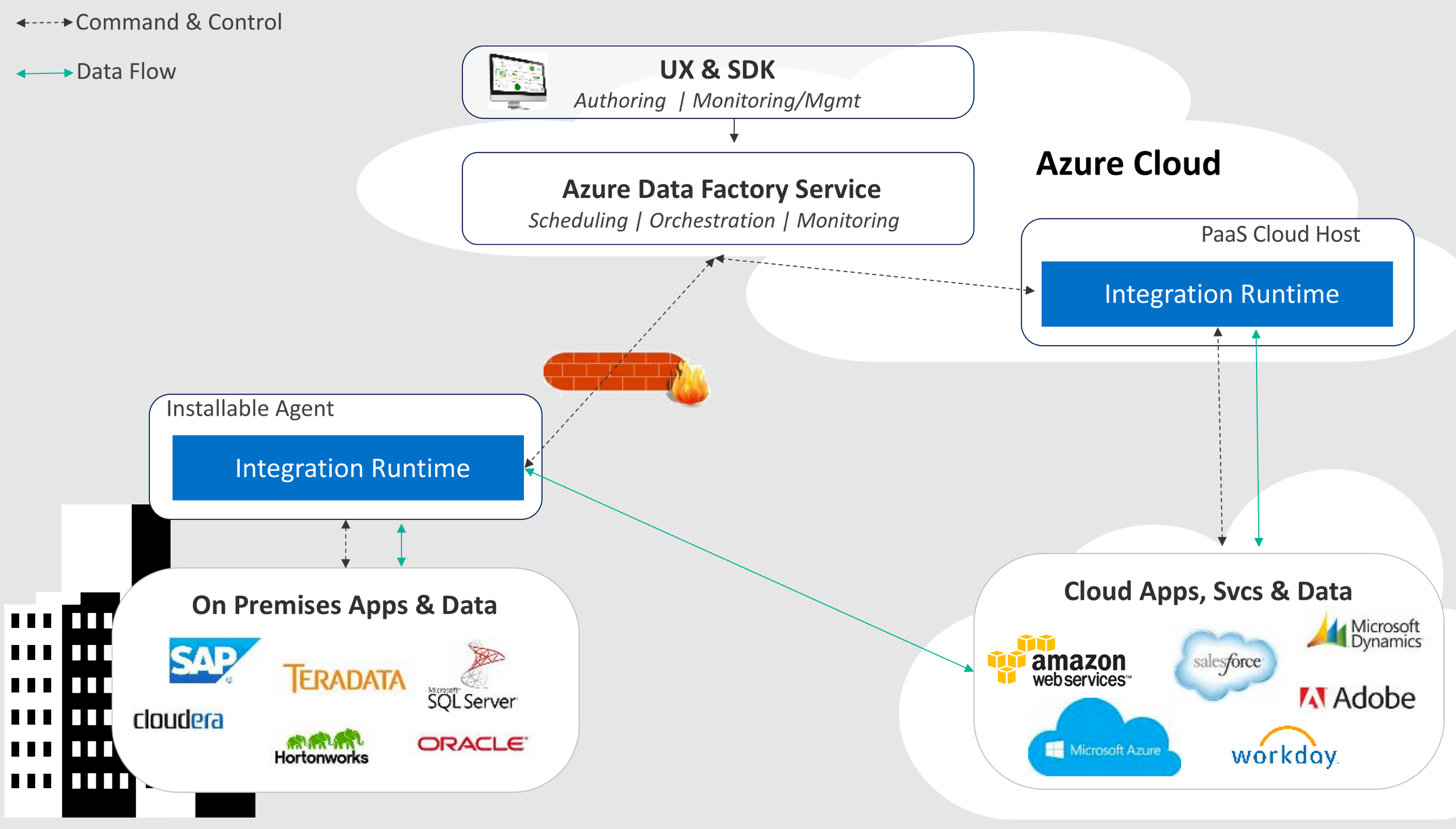
A key secondary function of the tool is the ability to connect to Azure Storage accounts including Blob storage containers. A SAS URL with time-based expiration access to a specific container can be shared to a team or person in order for them to upload their data that can then be picked up and used by Azure Data Factory. This is perfect for periodic uploads and for initiating projects with a large initial dataset.



# Azure Data Factory

Azure Data Factory is the primary tool for data movement and orchestration of data transformation in the data lake architecture. It is designed for operations at large scale with the ability to shuttle terabytes around in the environment on demand.

The core of this functionality is an architecture built around data pipelines that are managed in the cloud but executed inside integration runtimes.



The integration runtime hosts connectors that pull and push data to a variety of data sources and can be hosted either on premise or in the Azure cloud.

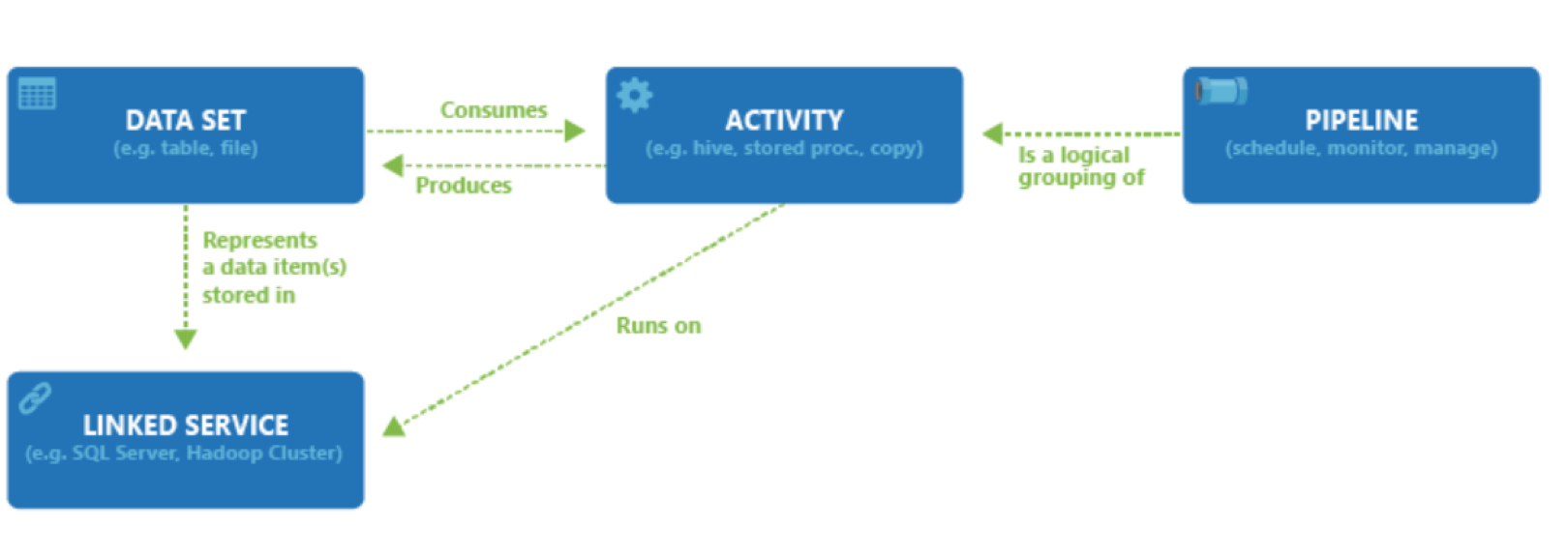
The architecture leverages integration runtimes hosted in the data center in order to access source like SAP BW, SAP Hana, Oracle databases for JDA, and on premises file shares.

Only the centralized ADF instance will have access to this integration runtime and the on premises data sources so that the /RAW folders in ADLS can be loaded.

Analytics data projects will get their own instances of Azure Data Factory and integration runtimes hosted in Azure with Azure only connectivity to support their data movement needs between ADLS lake content and their Azure cloud resources.

## Azure Data Factory artifacts

Azure Data Factory deployment artifacts are comprised of Pipelines, Activities, Data Sets and Linked Services.



### Pipelines

Container for the Activity and Data Set elements and the item of execution in ADF. Pipelines can be developed with a variety of activity types that can execute in sequence or in parallel and including flow control for looping and branching and invocation of other pipelines for modularity.

### Activity

The actions inside of a pipeline with data movement, data transformation, and control activities. The workhorse data movement tool is the Copy Data activity. There are data transformation for Azure Databricks for ELT transformation using a notebook. The control flow activities allow for data lookups, looping, branching and other programmatic activities.

### Data Set

Contain Details of the actual table or folder or source location of the data that is being moved and also contain the schema details for a data source that is structured like a SQL database table or CSV file that is being parsed. Can also be untyped for recursive binary copy operations from a folder to a folder.

## Azure Data Factory naming conventions

Azure Data Factory naming is critical to successfully managing a large number of artifacts in an ADF instance without collision or confusion. This applies to scenarios creating artifacts using the ADF UI or using PowerShell scripting via JSON where each file represents an artifact in ADF.

The naming convention begins with the use of uppercase names and underscores for spaces as a word separator. Suffixes will be used to assist with when similarly named artifacts that represent a logical entity like a table in a database could point to different end systems like Oracle database, file shares or Azure Data Lake Store as well as the type of the artifact itself.

## Artifact suffixes

|  |  |
| --- | --- |
| **Prefix** | **Connector** |
| ACT | Activity |
| DS | Dataset |
| IR | Integration Runtime |
| PIPE | Pipeline |

## Connector suffixes

|  |  |
| --- | --- |
| **Prefix** | **Connector** |
| ADL | Azure Data Lake Store |
| BLOB | Azure Blob Storage |
| FILE | File system |
| DB2 | DB2 |
| FTP | FTP |
| HDFS | HDFS |
| HTTP | HTTP |
| KV | Azure Key Vault |
| MONGO | MongoDB |
| ODBC | ODBC |
| ODATA | ODATA |
| ORACLE | Oracle Database |
| SAPHANA | SAP Hana |
| S3 | Amazon S3 |
| SAPBW | SAP BW |
| SFTP | SFTP |
| SQLDB | Azure SQL Database |
| SQLDW | Azure SQL Data Warehouse |
| SQLSRV | SQL Server |
|  | add more??? |

## Integration runtime naming

Integration runtime names reflect the name of the analytics solution that they are deployed. The will use the suffix of IR to identify them as an integration runtime. **Since the IR names are used for machine names internally inside of Azure dashes will be used instead of underscores in naming as ADF will prevent the use of an underscore in an IR name**.

Example for the Orion Reporting project:

ORION-REPORTING-IR

## Linked service naming

Linked services should be named directly for the server and database or data source that they point to and include a standard prefix for the type of data source they represent.

{system}\_{data source}\_{connector type}\_LS

|  |  |
| --- | --- |
| **Prefix** | **Connector** |
| System | Name of system being linked like JDA or SAP |
| Data source | Name of server and database |
| Connector type | Type of ADF connector |

Example for the Oracle JDA server VMWL5166 hosting database DB1970M:

JDA\_VMWL5166\_DB1970M\_ORACLE\_LS

Example for Azure Data Lake Store named Analytics:

\_ANALYTICS\_ADLS\_LS

## Pipeline naming

Pipelines will be named according to the data task that they focus on. For /RAW data ingestion the names will focus on the system and data source that is being processed including the necessary details in its naming. A /RAW ingestion pattern of having a single pipeline with a single copy activity for a single table of a data source will be leveraged as some data sources like SAP have hundreds of tables and supports easier creation of them via automated scripts and debugging failed pipelines.

{system}\_{data source}\_{connector type}\_PIPE

|  |  |
| --- | --- |
| **Prefix** | **Connector** |
| System | Name of system being linked like JDA or SAP |
| Data source | Name of server and database |
| Connector type | Type of ADF connector |

Example for the movement of data from raw ingestion of all the JDA FCST table from the VMWL5166 server which hosts the database DB1970M:

JDA\_VMWL5166\_DB1970M\_FCST\_PIPE

For analytics solutions the name will typically be more task and business focused on the ELT process being executed and there will be opportunities to put more than one copy activity and have a lot more complexity in their ADF pipelines.

## Data set naming

Data sets will be the most numerous artifacts in the system and good naming is essential to browse them from the UI or find them via a script that lists them.

{system}\_{data source}\_{table}\_{connector type}\_DS

|  |  |
| --- | --- |
| **Prefix** | **Connector** |
| System | Name of system being linked like JDA or SAP |
| Data source | Name of server and database |
| Table | Table inside of database |
| Connector type | Type of ADF connector |

Example for dataset name for JDA SCP system Oracle database DB1790M FCST table:

JDA\_SCP\_VMWL5166\_DB1790M\_FCST\_ORACLE\_DS

Example for Azure Data Lake Store dataset for the above JDA SCP Oracle database FCST table ingestion into RAW:

JDA\_SCP\_VMWL5166\_DB1790M\_FCST\_ADLS\_DS

## Linked services and Azure Key Vault

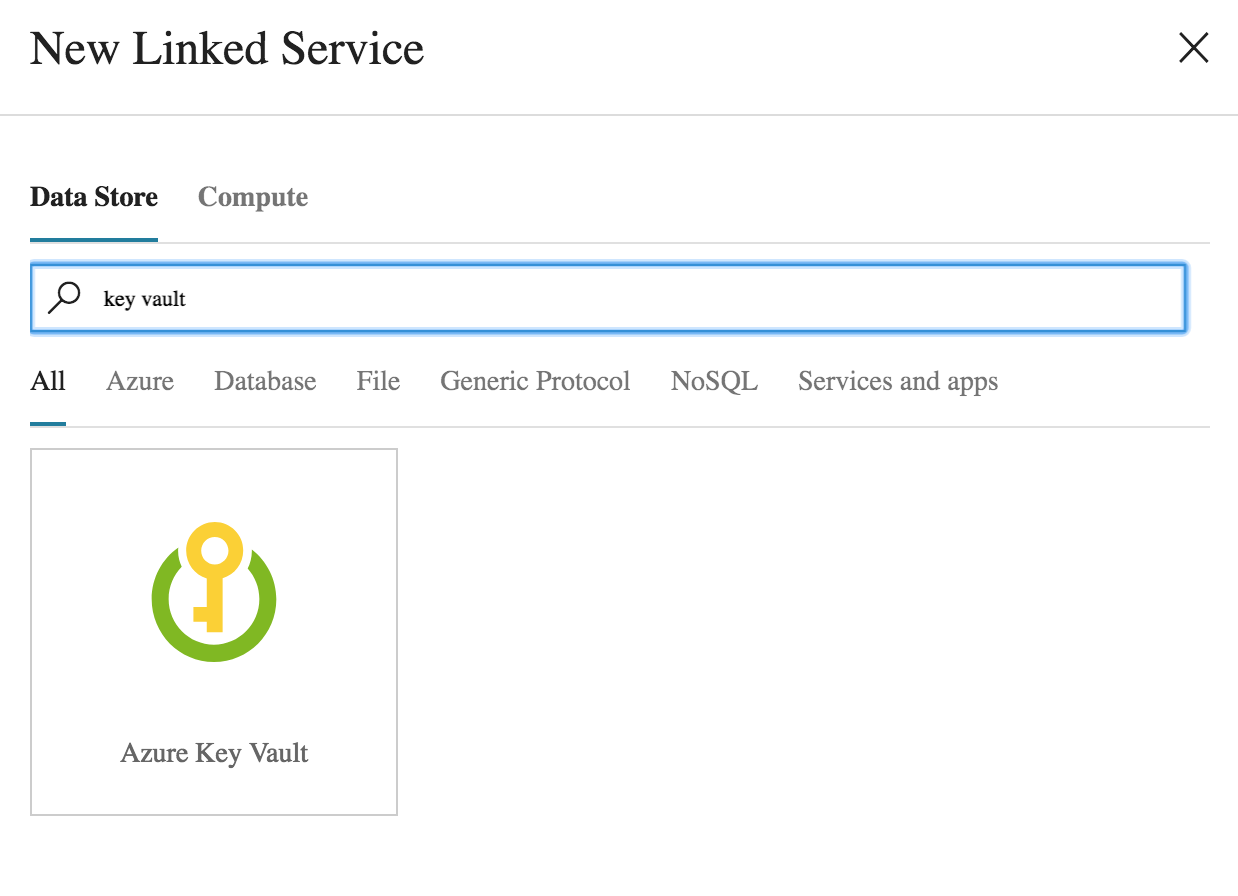
Data sources all require credentials whether username and password for a database or file share, an access key for Azure Blob Storage, or the service principal and key for Azure Data Lake Store. Azure Key Vault should be used as the storage area for all such secrets and should be referenced by Linked Services in their configuration.

The linked services should put a key in Azure Key Vault and name it by taking their normal name and appending a \_SECRET suffix at the end. An example for the Oracle JDA server VMWL5166 hosting database DB1970M would be:

JDA\_VMWL5166\_DB1970M\_ORACLE\_LS\_SECRET

## Azure Key Vault linked service setup

There is a setup requirement to create an Azure Key Vault Linked Service that can retrieve secrets from the proper Azure Key Vault on behalf of all ADF linked services that wish to use this security function. The process of creating the linked service begins with selection of Key Vault as the type of linked service:



After created the linked service to Azure Key Vault, take note of the service identity application ID information at the bottom of the dialog:

This service identity application ID is the managed service identity that represents your ADF instance and can be used to setup access policies in Azure Key Vault. To set up a new policy go to your instance of Azure Key Vault in the portal and click on the Access Policies link.

This show the current policies and new ones are created using Add New.

In the Select principal link type in the Service Application ID from the ADF linked service dialog displayed earlier to look up the correct identity for your ADF instance. On the left select just Secret GET permissions and hit OK to add the new policy.

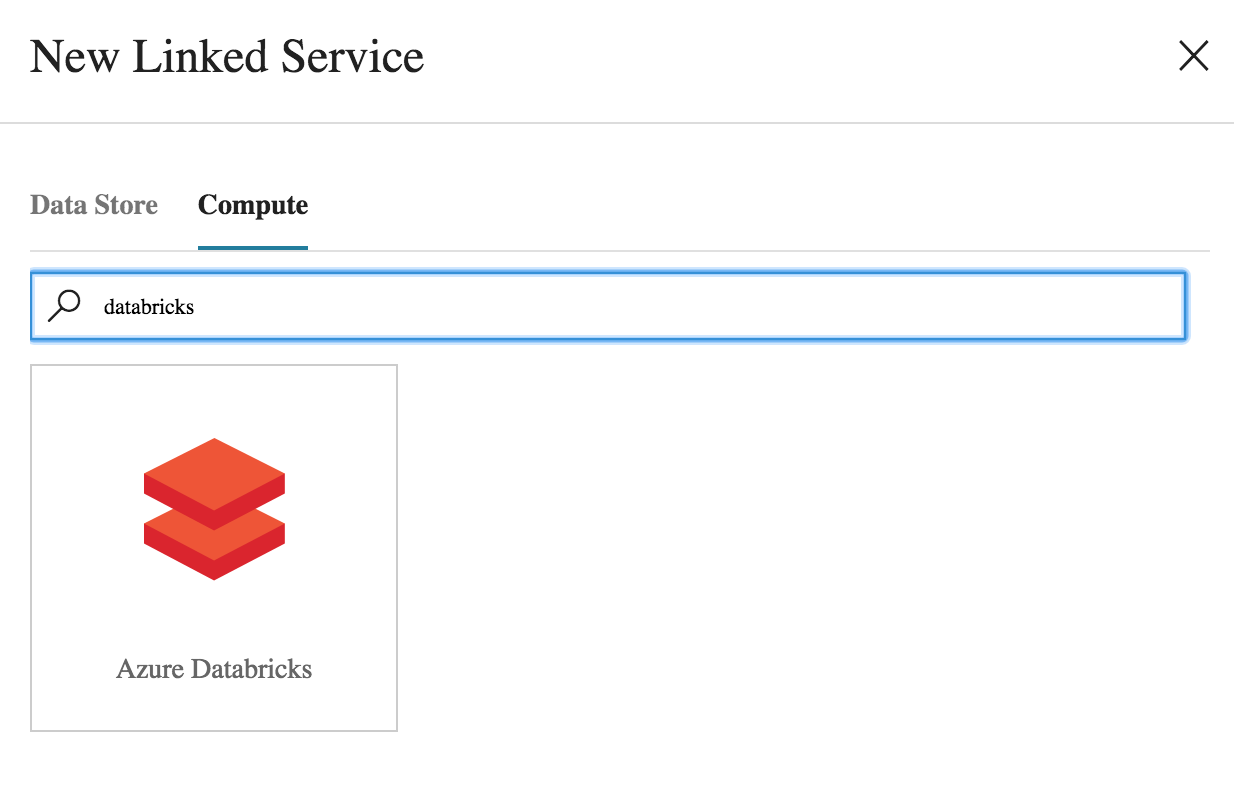
## ADF Key Vault secret selection in Linked Services

This can be done in JSON or in the UI for a linked service:

Most of the linked services just store and reference a password for the secret but some require a complete connection string for things like Azure SQL Database SQL authentication or for Azure Blob Storage. Refer to the Azure Data Factory Connectors documentation for the appropriate syntax.

## Azure Databricks linked service setup

A linked service is required to be able to invoke Azure Databricks notebooks for ELT compute processing from Azure Data Factory. Azure Databricks is under the Compute area for new linked services in ADF.



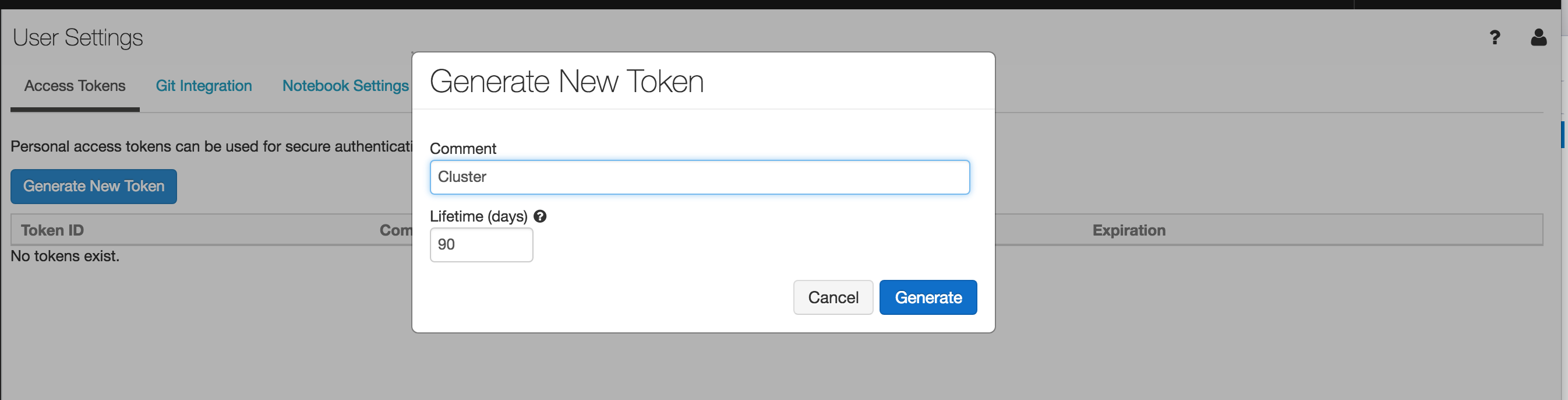
The creation dialog allow for two ways of running notebooks in a Job in Azure Databricks:

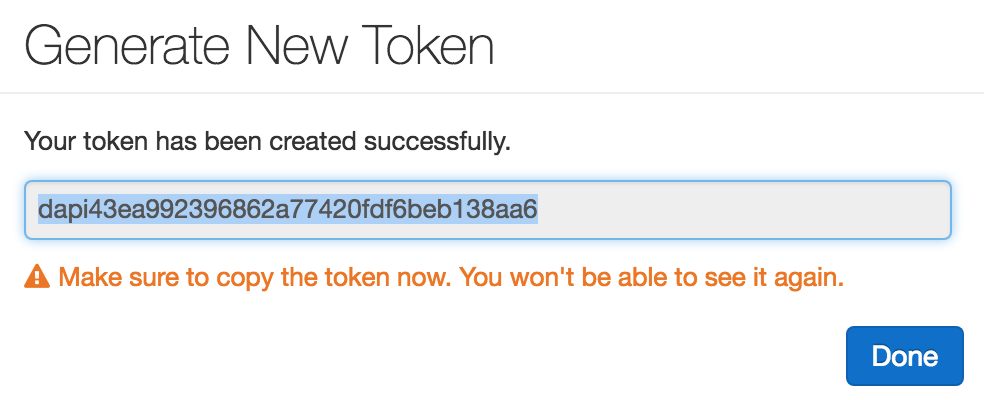
* New Job Cluster
* Existing Job Cluster

### Linked Service for existing Azure Databricks Cluster

For an existing cluster you need an access token from the Azure Databricks token and the cluster Id. The token is create from the User Settings menu selection when you click on the person icon on the right of the Azure Databricks console.

The User Settings page has an Access Tokens table and a Generate New Token button that is used to create tokens. The user doing this of course must have access to Create Cluster permissions in Azure Databricks.



The generate token dialog requires a comment to describe what the token is for and a Lifetime. After Generate is clicked the user has a one time opportunity to copy down this token. 

The second item needed for an existing cluster is the cluster Id. The Azure Databricks console has a Cluster button on the left hand side that navigates to the Clusters list.

Clicking on any cluster brings up its details and there is a Tags section at the very bottom that allows you to retrieve the cluster Id.

### Linked Service for new Azure Databricks Cluster

New cluster require the same access token from the existing cluster setup described above and a lot of additional information on cluster sizing and termination and other things described in detail in the Azure Databricks section of the document.

## Azure Data Factory CSV output format

Solutions have freedom to use their own formats, but the /RAW ingestion format will favor creating CSV files with ‘|’ pipes as the column separator, a blank value for nulls, and no quote characters for string delimiters.

## Timestamps and pipeline execution

Pipeline execution time should be the time source for all time stamps of output. The expression language in ADF easily enables this for the prescribed year/month/day format for folder part of a dataset path:

/RAW/JDA/VMWL5166/DB1970M/ /@{formatDateTime(utcnow(),'yyyy')}/@{formatDateTime(utcnow(),'MM')}/@{formatDateTime(utcnow(),'dd')}

## Azure Data Factory IAM roles

The Azure Data Factory Contributor IAM role must be granted to all project personnel at the Resource Group level in order to author, modify, and update all ADF artifacts with their ADF instance.

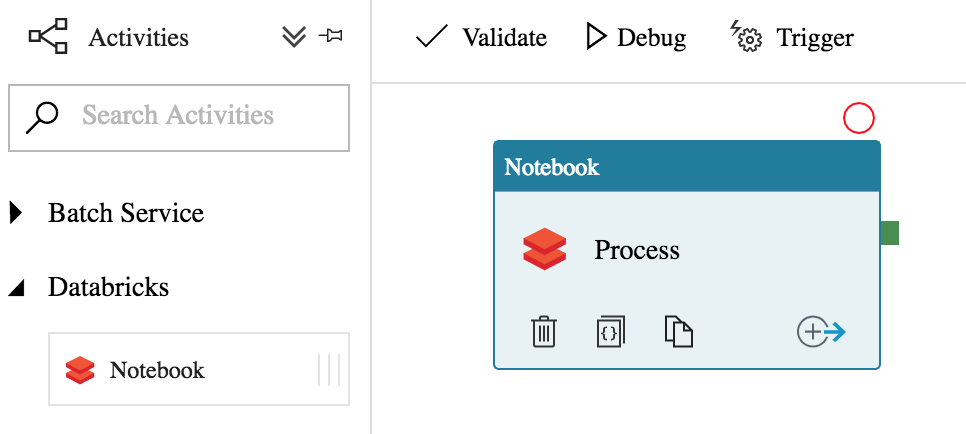
# Azure Databricks and ELT

Azure Databricks is a platform service in Azure that provides the ability to run Apache Spark workloads in an ELT or a data analytics perspective. This is the recommended ELT and data processing system.

Put more info here

The core of Azure Databricks are Spark clusters that run Spark jobs on demand from Databricks notebooks with either Python, Scala, or SQL code. Python and SQL are the recommended Notebook languages due to their widespread use in the data community.

Azure Databricks notebooks are directly callable from ADF pipelines via the Databricks pipeline activity.



## Notebook organization and file conventions

Standard method of organizing solution folder/scripts in Databricks

## Databricks Azure Data Lake Store mount points

Databricks supports permanent mount point to its internal DBFS filesystem via the dbutils library for remote resources like Azure Data Lake Store. The approach is recommended because it is persistent once created, abstracts the detailed URL and environment (dev or prod) of the lake from scripts, and it removes the need to lookup security credentials in each notebook session.

The recommended mount point namespace uses the following template:

/mnt/{top level folder path to mount in the lake}

Each project needs to be able mount the /RAW, /PROCESS, and /OUTPUT areas so three basic mounts are provided for each solution. For the ORION\_REPORTING solution it would look like the following:

/mnt/analytics/RAW

/mnt/analytics/PROCESS/ORION\_REPORTING

/mnt/analytics/OUTPUT/ORION\_REPORTING

These paths should can be used in all solution Python scripts for processing. The dbutils library also has an ls command for browsing files along with other directory and file related manipulation commands.

## Spark schema inference

Spark can infer the schema of flat files that don’t provide schema information like flat files and this feature is incredibly useful for experimenting and developing data pipelines.

<python code inferring schema>

Projects must remove with their usage of this feature and provide explicit schemas when moving to production as it can be a performance issue to scan files ahead of time for type definitions and could subtly cause errors when there is a change in the format of source data.

<python code with schema>

## Spark save folders and metadata file cleanup

Spark is a distributed processing framework that splits its operations into multiple distributed tasks including saving output to the Azure Data Lake Store.

<save command>

The dataframe save method ultimately results in a folder whose contents include files for each of the partitions of the dataframe but named by the original filename.

<picture of save output>

Spark can easily read these files from this folder on a dataframe read for additional processing but tools like ADF may be upset with the \_ metdata files that are created that validate the success of the operation.

Its rather trivial to run a dbutils command on that file directory to clean it up and have it only contain CSV files.

<command>

<output picture>

## Databricks Databases and Tables

Spark has rich functionality via Spark SQL to process dataframes using a SQL compliant syntax. This functionality is immediately available for any dataframe that calls the dataframe createOrReplaceTempView method:

<example>

This is a temporary table that is only available in the current session. One can immediately use a SQL syntax against that table named from the call:

<example>

Spark also has the concept of a catalog which stores metadata about databases, tables, and schema that is long lived and persistent across sessions and across the cluster. **These are not recommended** to maximize portability and deployment from development to production environments for ELT scripts.

## Spark query performance and optimization

The performance of a data pipeline in Spark is affected by many factors including but not limited to the size of the files (large or small), their format (CSV versus Parquet), compression (can it be split or not), the type of filters, transformations, aggregates or joins being processed against them, and the number of nodes and node size in the compute cluster.

Joins in particular are the biggest determining factor in most pipeline jobs that are aggregating information. Distributed systems pay a huge price in performance when data on one node in a cluster has to travel to one or a host of other nodes in that cluster via the network in a shuffle operation.

It is highly recommended that solutions use the /PROCESS area to import /RAW data from text and then organize according to their own data in using optimized file formats like Parquet and partitioning to greatly enhance the performance of their data pipelines.

The best optimization practices start with correct file distribution and partitioning and leverage tools like Parquet column compression in order to achieve orders of magnitude query performance improvements without incurring the costs of a larger cluster operation.

# Azure SQL Database

Azure SQL Database is a fully managed PaaS service in Azure that provides all the relational features of the SQL Server database system in the cloud. This includes data encryption at rest, point in time restore for the last 30 days, row level security support, and authentication via user managed passwords or integration with Azure AD security.

This is the primary tool used by analytics projects as a source for their reporting tool queries by Tableau or Power BI. They will use the functionality of Azure Data Factory pipelines in conjunction with any other ETL/ELT tooling to load this database from the sources that are available in the /RAW folders of the Azure Data Lake Store.

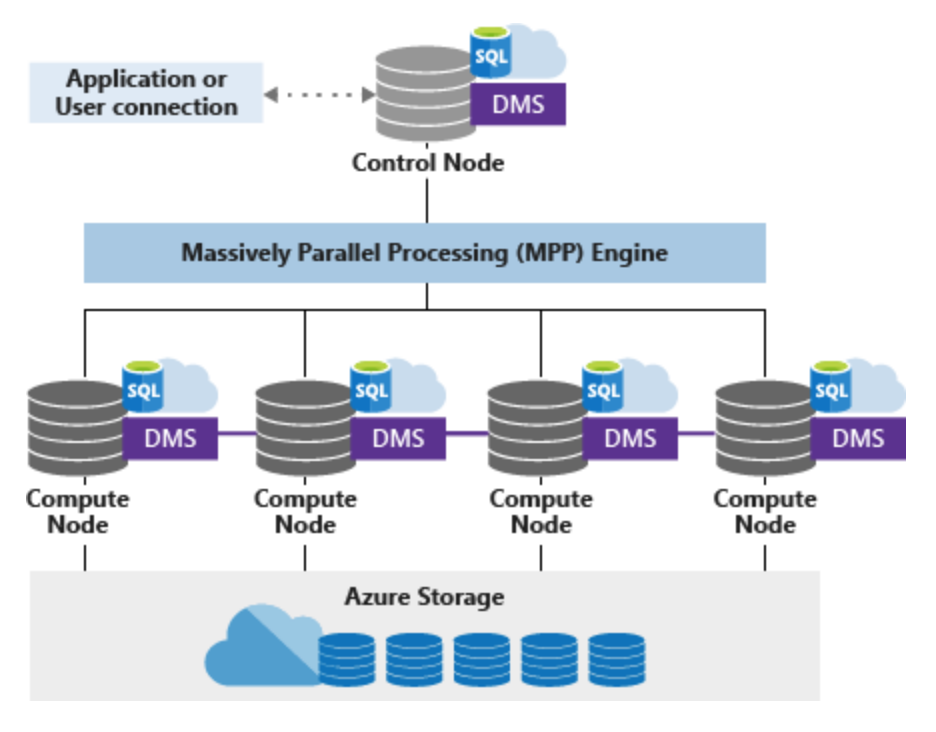
This puts the query load on a simpler and more cost effective relational system and lets analytics platforms like Databricks or HDInsight Hadoop clusters be used for creating the datasets to be put into the Azure SQL Database.

A limited number of logical Azure SQL Servers will be provisioned and Azure SQL Databases will be assigned to solutions teams with all security settings for access for that solution configured at the database with the idea of “contained database” security.

Diagram of logical SQL servers and database and security

# Azure SQL Data Warehouse

Azure SQL Data Warehouse is a massively parallel processing system designed for sub second query response on large datasets. It supports the normal functionality of SQL Server but has a radically different architecture which requires data profiling and planning to ensure its distributed storage and query processing system is efficiently allocated and balanced for its query workloads.



This system will only be deployed for specialized projects that have had an architecture review as there is a significantly higher cost for this system over SQL Database and can perform inefficiently if not properly configured. This is also not a system designed for a large number of users to query (maximum is 16 depending on version deployed) at one time so should be treated more as an advanced analytical processing engine similar to a large Databricks/Spark cluster.

# Azure Data Catalog

Azure Data Catalog is the primary metadata repository for data sources in the lake architecture.

Picture of Azure Data Catalog

The analytics projects will be the primary content creators with the Azure Data Catalog as they will be researching their data sources for information.

# Dev and Prod Environments

The data lake architecture utilizes separate environments for development and production deployments to ensure the stability of solutions for business customers while providing a development sandbox to build and test solutions.

Picture of dev and test environments

Automated deployment tools and source code control will be used with all Azure and solution technologies to ensure that there is an easy path of migration of resources from development to production with easy rollback on errors or issues. The two environments will use the same folder naming conventions to ease movement of ADF pipelines and Azure Databricks scripts via simple configuration changes.

The following resources will be deployed to support development and production for /RAW ingestion and Vault processing:

Production

|  |  |
| --- | --- |
| Name | Resource Type |
| analyticsprodadls  analyticsdevadls  vaultprodadls  vaultdevadls | Azure Data Lake Store |
| analyticsprodakv  analyticsdevakv  vaultprodakv  vaultdevakv | Azure Key Vault |
| analyticsprodadf  analyticsdevadf  vaultprodadf  vaultdevadf | Azure Data Factory |
| analyticsprodadb  analyticsdevadb  vaultprodadb  vaultdevadb | Azure Data Bricks |
| analtyicsprodsqlsvr  analyticsdevsqlsvr | Azure SQL Server |

Each solution will have a provisioned development and production resources for Azure Data Factory, Azure Databricks, and Azure SQL Database. They will leverage the Azure Key Vault and Azure Data Lake Store and Azure SQL Server of the core resources maintained centrally.

# Azure Subscriptions and Resource Groups

The Azure infrastructure provides management, billing, and security boundaries in different ways across an enterprise customer’s Azure EA enrollment. All billing and payments are centralized at the Azure EA enrollment level with tools to allow assignment of privileges to creation Azure subscriptions and get access to full billing reports at the enterprise level.

Subscriptions in Azure are the top-level container for resources in Azure and the security and billing associated with them. All Azure resources like Azure Data Lake Store, Azure Data Factory, Azure Databricks, Azure VMs and a whole host of other technologies are always created inside of an Azure subscription. Users can be given privileges via IAM roles at the Azure subscription level, so they can manage any resource inside of it. Billing is most easily rolled up and reported at the entire Azure subscription level and as such is used for this purpose as well by creating subscriptions for teams to manage their own Azure resources.

Resource Groups are containers inside of an Azure subscription used for deployment, delegated access control and billing. All Azure resources must be created in inside of an Azure resource group and the deployment functionality of the ARM template system is centered on them. Users can be assigned IAM roles at the Azure resource group level where they get applied to all resources inside. Billing reports always include what resource groups are involved with consumption and carry the tags applied at the Azure resource group level, so this provides a great project specific or team specific way of tracking spend.

There are many different patterns for delegating Azure management, but two models predominate in how resources are doled out to teams for consumption: Azure Subscriptions and Azure Resource Groups.

<model of subscription versus resource group>

The core infrastructure that makes up the Azure Data Lake and /RAW ingestion machinery will be deployed in a separately managed Azure subscription. The importance of isolating access control to the Azure Data Lake Store accounts and the /RAW Azure Data Factory engines cannot be overstated as to of their importance in the entire Data Lake Architecture.

For larger solutions like BI Factory it makes sense to provide them with more autonomy and control and responsibility with managing their own Azure Subscription resource groups. Even with this model, Azure resource groups should be used to subdivide and manage what presumably will be a complex set of solution deployments

For smaller projects that may be temporary or part of an MVP, a shared Azure subscription will be used with each project being given an Azure Resource Group with their solution artifacts. They will also be restricted from creating new Azure resources and will be given a templated supported Azure deployment to build their solutions.