# Azure Data Lake & Best Practices

## Introduction Azure Data Lake store:

The Azure Data Lake Store is an integral component for creating a data lake in Azure as it is where data is physically stored.

Azure Data Lake enables you to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics.

Azure Data Lake Store can be accessed from Hadoop (available with HDInsight cluster) using the WebHDFS-compatible REST APIs. It is specifically designed to enable analytics on the stored data and is tuned for performance for data analytics scenarios. Out of the box, it includes all the enterprise-grade capabilities—security, manageability, scalability, reliability, and availability—essential for real-world enterprise use cases.

## Key capabilities of the Azure Data Lake:

### Built for Hadoop

The Azure Data Lake store is an Apache Hadoop file system compatible with Hadoop Distributed File System (HDFS) and works with the Hadoop ecosystem.

Data stored in Azure Data Lake Store can be easily analyzed using Hadoop analytic frameworks such as MapReduce or Hive. Microsoft Azure HDInsight clusters can be provisioned and configured to directly access data stored in Data Lake Store. Files are split up and distributed across an array of cheap storage.

### Unlimited storage, petabyte files:

Azure Data Lake Store provides unlimited storage and is suitable for storing a variety of data for analytics. It does not impose any limits on account sizes, file sizes, or the amount of data that can be stored in a data lake. Individual files can range from kilobyte to petabytes in size making it a great choice to store any type of data. Data is stored durably by making multiple copies and there is no limit on the duration of time for which the data can be stored in the data lake.

### Performance-tuned for big data analytics

Azure Data Lake Store is built for running large scale analytic systems that require massive throughput to query and analyze large amounts of data. The data lake spreads parts of a file over a number of individual storage servers. This improves the read throughput when reading the file in parallel for performing data analytics.

### Enterprise-ready: Highly-available and secure

Azure Data Lake Store provides industry-standard availability and reliability. Your data assets are stored durably by making redundant copies to guard against any unexpected failures.

Data Lake Store also provides enterprise-grade security for the stored data.

### All Data

Azure Data Lake Store can store any (structured, semi-structured and unstructured) data in their native format, as is, without requiring any prior transformations. Data Lake Store does not require a schema to be defined before the data is loaded, leaving it up to the individual analytic framework to interpret the data and define a schema at the time of the analysis.

Azure Data Lake Store containers for data are essentially folders and files. Data Lake Store does not perform any special handling of data based on the type of data it stores.

## Structure Data in Azure Data Lake store:

When data is stored in Data Lake Store, the file size, number of files and folder structure has an impact on performance. The following section describes **best practices** in these areas.

### File Size

Typically, analytics engines such as HDInsight and Azure Data Lake Analytics have a per-file overhead. If you store your data as many small files, this can negatively affect performance. In general, organize your data into larger sized files for better performance. As a rule of thumb, organize data sets in files of 256MB or larger. In some cases such as images and binary data, it is not possible to process them in parallel. In these cases, it is recommended to keep individual files under 2GB. Sometimes, data pipelines have limited control over the raw data which has lots of small files. It is recommended to have a “cooking” process that generates larger files to use for downstream applications.

### Organizing Time Series data in folders

For Hive and ADLA workloads, partition pruning of time-series data can help some queries read only a subset of the data which improves performance. Those pipelines that ingest time-series data often place their files with a very structured naming for files and folders. Below is a very common example we see for data that is structured by date:

*\DataSet\YYYY\MM\DD\datafile\_YYYY\_MM\_DD.tsv*

Notice that the date time information appears both as folders and in the filename.

For date and time, the following is a common pattern:

*\DataSet\YYYY\MM\DD\HH\mm\datafile\_YYYY\_MM\_DD\_HH\_mm.tsv*

### Folder structure in Azure Data Lake:

Your number one goal in terms of how a data lake is architected and structured is that someone from your organization (may not necessarily have expertise in data per se) should still find the structure of your data lake, the names and the contents of the data set stored within it to be fairly self-explanatory.

That way, a self-service kind of approach can work. What you want to avoid is something that’s such a cluster of undocumented stuff that nobody really feels competent to go near it.

So, when setting up your Azure Data Lake Store you will want to initially create the following folders in your Root:

1. RAW – Always store a copy of raw input data as first rule of thumb.
2. Enriched (In Process) – Data is cleaned, validated, enriched and augmented with additional sources.
3. Curated (Processed) – Data ready for consumption by the business.

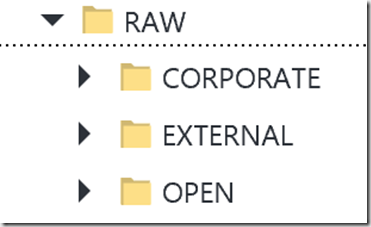
The Raw data area, the landing place for any files entering the lake, has sub-folders for each source of data. This allows for the easy browsing of the data sources within the Lake and ensures we are not receiving the same data twice, even if we use it within different systems.

The Enriched and Curated layers however, have a specific purpose in mind. We don’t take data and enrich/clean/process it without a business driver. We can therefore assign a project or system name to it, at this point it is organized into these end-systems. This means we can view the same structure within Enriched as within Curated.

Essentially Raw data is categorized by **Source** whilst Enriched and Curated data is categorized by **Destination**.

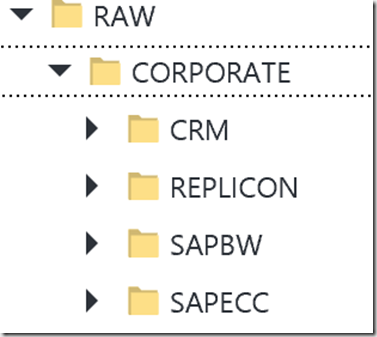
Example as shown below:

Raw is where data is landed in directly from source and the underlying structure will be organized ultimately by Source.

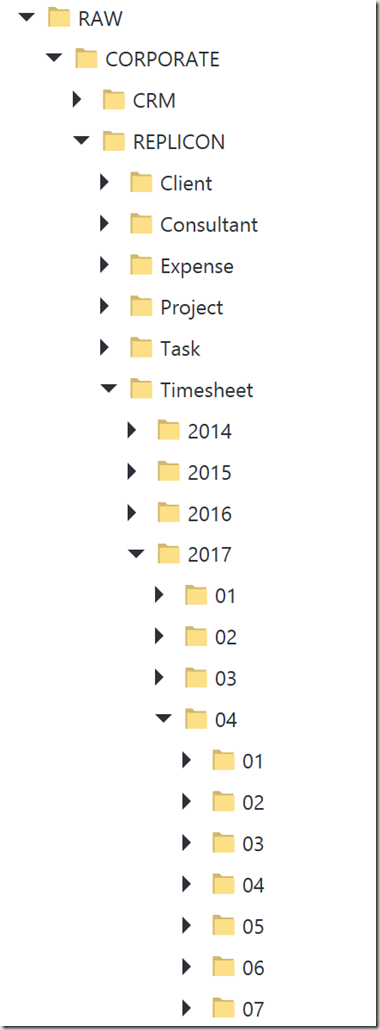
[](http://blogs.adatis.co.uk/ustoldfield/image.axd?picture=image_1.png)

Source is categorized by Source Type, which reflects the ultimate source of data.

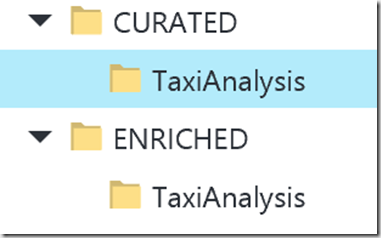
Within the Source Type, data is further organized by Source System.

[](http://blogs.adatis.co.uk/ustoldfield/image.axd?picture=image_2.png)

Within the Source System, the folders are organized by Entity and, if possible, further partitioned using the standard Azure Data Factory Partitioning Pattern of Year > Month > Day etc., as this will allow you to achieve partition elimination using file sets.

[](http://blogs.adatis.co.uk/ustoldfield/image.axd?picture=image_3.png)

The folder structure of Enriched and Curated is organized by Destination Data Model. Within each Destination Data Model folder is structured by Destination Entity.

[](http://blogs.adatis.co.uk/ustoldfield/image.axd?picture=image_4.png)

To summarize, structure in your Azure Data Lake Store is key to maintaining order:

• You need to enforce and maintain folder structure.

• Remember that structure is necessary whether using unstructured data or structured data & SQL.

• Bear in mind that schema on read applies temporary structure.

# Azure Data Factory Best Practices

## Naming conventions

The following table provides naming rules for Data Factory artifacts.

| Name | Name Uniqueness | Validation Checks |
| --- | --- | --- |
| Data Factory | Unique across Microsoft Azure. Names are case-insensitive, that is, MyDF and mydf refer to the same data factory. | * Each data factory is tied to exactly one Azure subscription. * Object names must start with a letter or a number, and can contain only letters, numbers, and the dash (-) character. * Every dash (-) character must be immediately preceded and followed by a letter or a number. Consecutive dashes are not permitted in container names. * Name can be 3-63 characters long. |
| Linked Services/Tables/Pipelines | Unique with in a data factory. Names are case-insensitive. | * Maximum number of characters in a table name: 260. * Object names must start with a letter, number, or an underscore (\_). * Following characters are not allowed: “.”, “+”, “?”, “/”, “<”, ”>”,”\*”,”%”,”&”,”:”,”\” |
| Resource Group | Unique across Microsoft Azure. Names are case-insensitive. | * Maximum number of characters: 1000. * Name can contain letters, digits, and the following characters: “-”, “\_”, “,” and “.” |

### Linked services & Datasets:

When it comes to deciding on a naming convention there is not a ‘one size fits all’ scenario. 

Below table can be used as a reference to name Linked services and Datasets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type** | **Linked Service** | **Name** | **Linked Service** | **Dataset** | **Full** |
| Azure | [Azure Blob storage](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-blob-connector) | ABLB\_ | LS\_ABLB\_ | DS\_ABLB\_ | LS\_ABLB\_Example |
|  | [Azure Data Lake Store](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-datalake-connector) | ADLS\_ | LS\_ADLS\_ | DS\_ADLS\_ | LS\_ADLS\_Example |
|  | [Azure SQL Database](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-sql-connector) | ASQL\_ | LS\_ASQL\_ | DS\_ASQL\_ | LS\_ASQL\_Example |
|  | [Azure SQL Data Warehouse](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-sql-data-warehouse-connector) | ASDW\_ | LS\_ASDW\_ | DS\_ASDW\_ | LS\_ASDW\_Example |
|  | [Azure Table storage](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-table-connector) | ATBL\_ | LS\_ATBL\_ | DS\_ATBL\_ | LS\_ATBL\_Example |
|  | [Azure DocumentDB](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-documentdb-connector) | ADOC\_ | LS\_ADOC\_ | DS\_ADOC\_ | LS\_ADOC\_Example |
|  | [Azure Search Index](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-azure-search-connector) | ASER\_ | LS\_ASER\_ | DS\_ASER\_ | LS\_ASER\_Example |
| Databases | [SQL Server\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-sqlserver-connector) | MSQL\_ | LS\_SQL\_ | DS\_SQL\_ | LS\_SQL\_Example |
|  | [Oracle\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-oracle-connector) | ORAC\_ | LS\_ORAC\_ | DS\_ORAC\_ | LS\_ORAC\_Example |
|  | [MySQL\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-mysql-connector) | MYSQ\_ | LS\_MYSQ\_ | DS\_MYSQ\_ | LS\_MYSQ\_Example |
|  | [DB2\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-db2-connector) | DB2\_ | LS\_DB2\_ | DS\_DB2\_ | LS\_DB2\_Example |
|  | [Teradata\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-teradata-connector) | TDAT\_ | LS\_TDAT\_ | DS\_TDAT\_ | LS\_TDAT\_Example |
|  | [PostgreSQL\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-postgresql-connector) | PSQL\_ | LS\_PSQL\_ | DS\_PSQL\_ | LS\_PSQL\_Example |
|  | [Sybase\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-sybase-connector) | SYBA\_ | LS\_SYBA\_ | DS\_SYBA\_ | LS\_SYBA\_Example |
|  | [Cassandra\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-cassandra-connector) | CASS\_ | LS\_CASS\_ | DS\_CASS\_ | LS\_CASS\_Example |
|  | [MongoDB\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-on-premises-mongodb-connector) | MONG\_ | LS\_MONG\_ | DS\_MONG\_ | LS\_MONG\_Example |
|  | [Amazon Redshift](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-amazon-redshift-connector) | ARED\_ | LS\_ARED\_ | DS\_ARED\_ | LS\_ARED\_Example |
| File | [File System\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-onprem-file-system-connector) | FILE\_ | LS\_FILE\_ | DS\_FILE\_ | LS\_FILE\_Example |
|  | [HDFS\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-hdfs-connector) | HDFS\_ | LS\_HDFS\_ | DS\_HDFS\_ | LS\_HDFS\_Example |
|  | [Amazon S3](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-amazon-simple-storage-service-connector) | AMS3\_ | LS\_AMS3\_ | DS\_AMS3\_ | LS\_AMS3\_Example |
|  | [FTP](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-ftp-connector) | FTP\_ | LS\_FTP\_ | DS\_FTP\_ | LS\_FTP\_Example |
| Others | [Salesforce](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-salesforce-connector) | SAFC\_ | LS\_SAFC\_ | DS\_SAFC\_ | LS\_SAFC\_Example |
|  | [Generic ODBC\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-odbc-connector) | ODBC\_ | LS\_ODBC\_ | DS\_ODBC\_ | LS\_ODBC\_Example |
|  | [Generic OData](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-odata-connector) | ODAT\_ | LS\_ODAT\_ | DS\_ODAT\_ | LS\_ODAT\_Example |
|  | [Web Table (table from HTML)](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-web-table-connector) | WEBT\_ | LS\_WEBT\_ | DS\_WEBT\_ | LS\_WEBT\_Example |
|  | [GE Historian\*](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-odbc-connector#ge-historian-store) | GEHI\_ | LS\_GEHI\_ | DS\_GEHI\_ | LS\_GEHI\_Example |

### Pipelines

Pipelines are of two different types. The copy data activity and the data transformation activity.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Action** | **Example** |
| Data movement Activity | PL\_DATA\_ | NA | PL\_DATA\_DS\_SQL\_Person\_To\_DS\_ABLB\_Person |
| Data transformation pipeline | PL\_TRAN\_ | SPRC - Stored Procedure | PL\_TRAN\_SPRC\_CleanDimAccount |
|  | PL\_TRAN\_ | DNET - Script | PL\_TRAN\_DNET\_AggregateSales |
|  | PL\_TRAN\_ | ADLK - Azure Data Lake | PL\_TRAN\_ADLK\_AggregateSales |
|  | PL\_TRAN\_ | HIVE - Hive | PL\_TRAN\_HIVE\_AggregateSales |
|  | PL\_TRAN\_ | PIG - Pig | PL\_TRAN\_PIG\_AggregateSales |
|  | PL\_TRAN\_ | MAPR - MapReduce | PL\_TRAN\_MAPR\_AggregateSales |
|  | PL\_TRAN\_ | HADP - Hadoop Stream | PL\_TRAN\_HADP\_StreamData |
|  | PL\_TRAN\_ | AML - Azure Machine Learning | PL\_TRAN\_AML\_CalculateMonthlyChurn |

### Activities

The activities can be prefixed with the number to indicate the run order.

AC\_0\_TruncateCustomer

AC\_1\_LoadCustomer

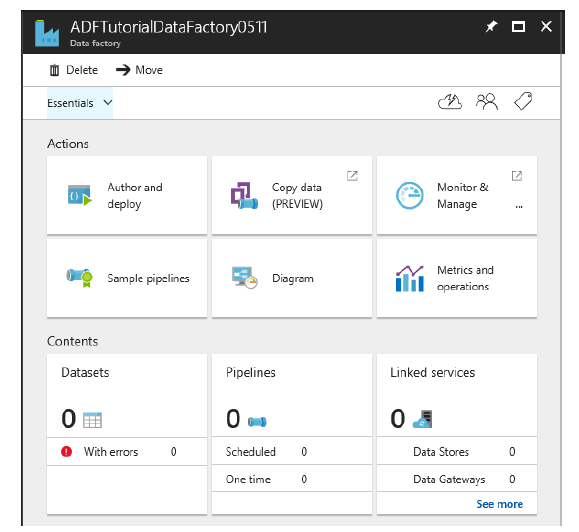
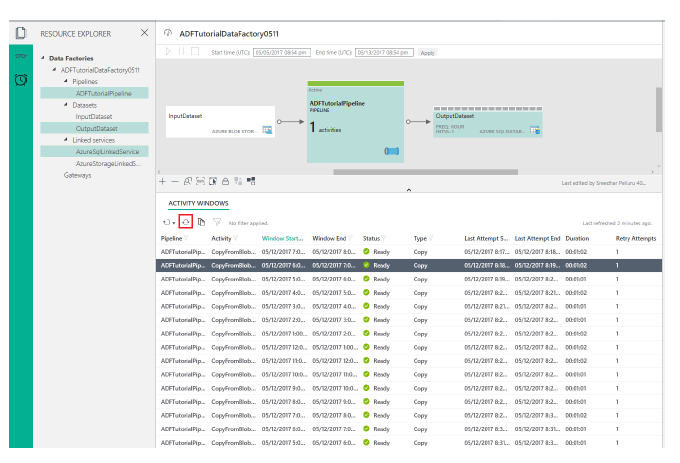
In summary, deciding on a naming convention that works for you is one of the most important considerations when it comes to using ADF.

## Monitor and Manage Pipelines:

In this step, you use the Azure portal to monitor what’s going on in an Azure data factory.

Azure Portal provides you user interface to monitor pipelines in your data factory by using the Monitor & Manage application:  
  
  
  
NOTE

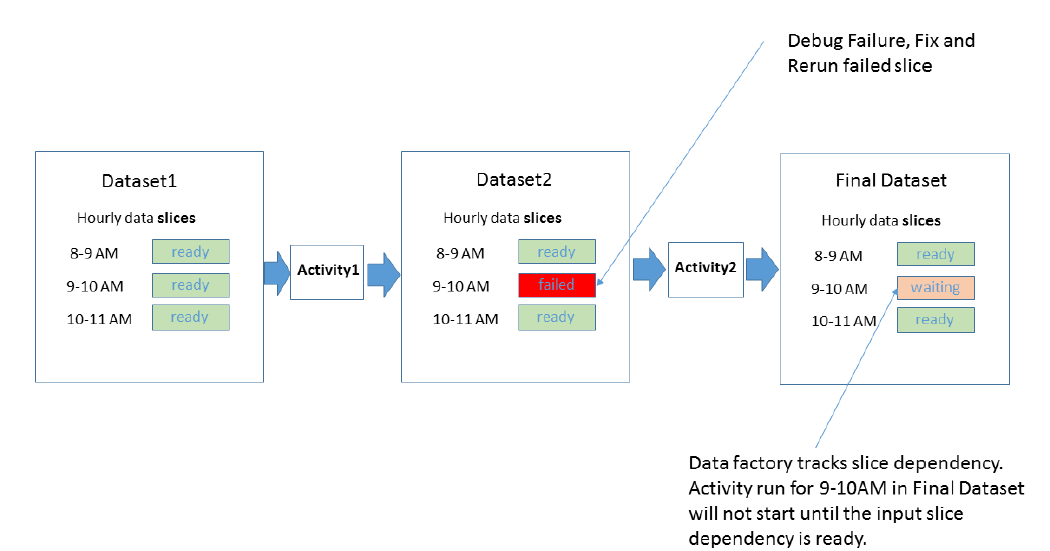
1. Click Monitor & Manage tile on the home page for your data factory.

  
  
  
  
2. You should see Monitor & Manage application in a separate tab  
  


### Rerun a failed data slice

*Each unit of data consumed or produced by an activity run is called a data slice*.

When an error occurs while processing a data slice, you can find out why the processing of a slice failed by using Azure portal blades or Monitor and Manage App. See Monitoring and managing pipelines using Azure portal blades or Monitoring and Management app for details. Consider the following example, which shows two activities Activity1 and Activity 2. Activity1 consumes a slice of Dataset1 and produces a slice of Dataset2, which is consumed as an input by Activity2 to produce a slice of the Final Dataset.



The diagram shows that out of three recent slices, there was a failure producing the 9-10 AM slice for Dataset2.Data Factory automatically tracks dependency for the time series dataset. As a result, it does not start the activity run for the 9-10 AM downstream slice.

Data Factory monitoring and management tools allow you to drill into the diagnostic logs for the failed slice to easily find the root cause for the issue and fix it. After you have fixed the issue, you can easily start the activity run to produce the failed slice. For more information on how to rerun and understand state transitions for data slices, see Monitoring and managing pipelines using Azure portal blades or monitoring and Management app.

After you rerun the 9-10 AM slice for Dataset2, Data Factory starts the run for the 9-10 AM dependent slice on the final dataset.



### Important Note

The Best practices will evolve over the time, as we actually work and understand the business use cases for CCH Data.

References:

<https://docs.microsoft.com/en-us/azure/data-lake-store/>

<http://blogs.adatis.co.uk/ustoldfield/post/Shaping-The-Lake-Data-Lake-Framework>

<http://blogs.adatis.co.uk/ustoldfield/post/Azure-Data-Lake-Store-Storage-and-Best-Practices>

<https://docs.microsoft.com/en-gb/azure/data-factory/v1/data-factory-naming-rules>

<http://blogs.adatis.co.uk/terrymccann/post/Azure-Data-Factory-Suggested-naming-conventions-and-best-practices>

<https://github.com/hareeshnagaraj/azure-content/blob/master/articles/data-factory/data-factory-copy-activity-performance.md>