ADLS (gen1) Architectural pattern for MLS using Datasets

Many organizations are now focusing on a single version of truth of their data, typically via some form of a Datalake strategy. This brings several benefits such as, single access point, less silos, and enriched dataset via amalgamation of data from various sources. Microsoft answer to this strategy is the Datalake store [1]. Datalake Store brings various benefit to enterprises, such as security, manageability, scalability, reliability as well as availability. Typical approach to Datalake strategy we see customers are using is the hierarchical approach (see fig 1), where the data is first ingested into a landing layer, typically referenced as the "Raw Datalake", data is then processed, filtered, optimized and placed in the "Curated Datalake". This is then further refined/processed based on the application/service specific logic which is referred to as the "Product Datalake".

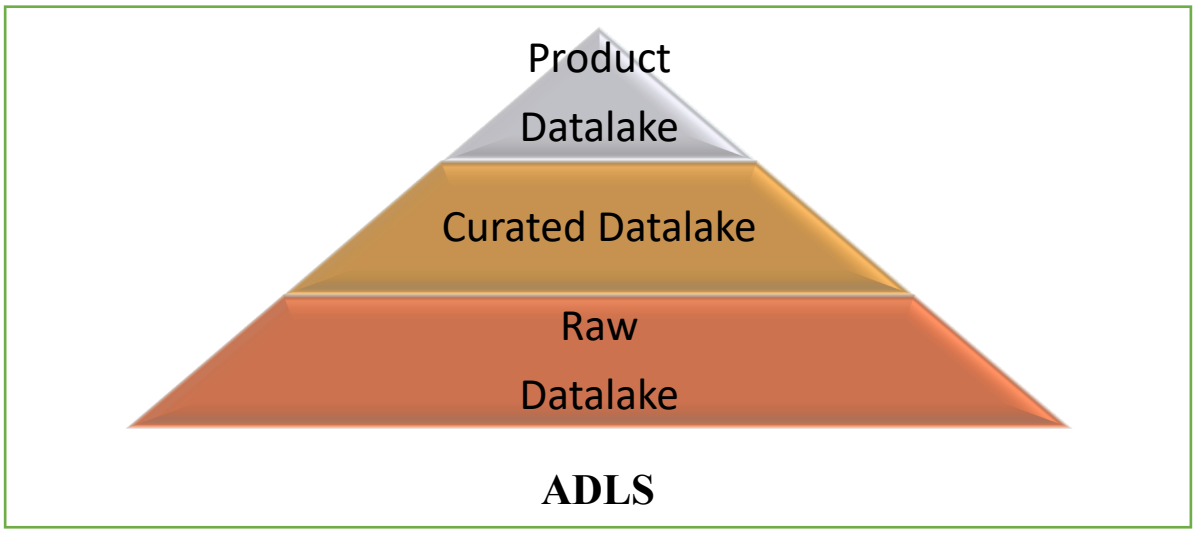

Fig 1- Datalake Hierarchal Strategy

Raw Datalake

A raw Datalake provides single version of truth for all the data in an organization, that can be seen as the landing zone for all the data. Data from various sources, whether it is structed, semi-structured and unstructured are ingested in its native format, which may not contain any optimization, some basic data quality check, such as total number of rows ingested, etc.
When data is landed, couple things to consider, who has access to this data, how is this data going to be process. This is where you would consider what the data partition strategy should be what file format and compression technique to consider. This layer is typically owned by the IT.

Curated Datalake

While the raw Datalake is targeted at the organization level, the curated Datalake is focused on the division or OU level. Each division may define its own business rules and constrains of how the data should be processed, presented and accessed. Data are typically stored in an

optimized format and are generally aggregated, filtered and cleaned (handling missing values, outliers and others). Curated data tend to have more structure and are grouped to specific domain, such as Finance, HR, etc.

Product Datalake

The product Datalake typically tend to application specific, and normally filtered to specific interest of the application. The application focus here are typically Machine learning based applications, which rely on large volume of data to use for modelling and providing batch or real-time inferencing and written back to the Product Datalake.

Such benefits would be futile, if you are not able to perform advance analytics in a way that gives the predictive and proscriptive capabilities. Here we have defined an Azure ML stack infused with Datalake strategy.

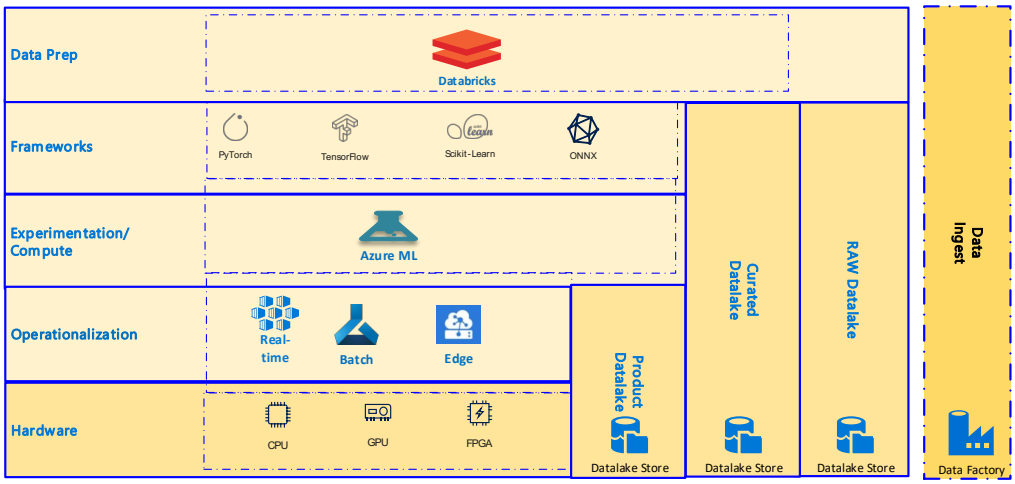

Figure 2 – Datalake infused in ML Stack.

The ML stack consists of various layers which are infused with different category of Datalake Store. The first layer "Data Prep" is where raw data is processed depending on the business logic and domain constraints, typically using Databricks, and is made available to product specific Datalake.
Layer two consists of various frameworks to be used for ML problem at hand and would typically use the Product Datalake to conduct various feature engineering.

Layer three consists of Azure Machine learning, for carryout experiments using various compute targets, tracking of experiments, and would use the product Datalake as the source of experimentation data, and any writeback would take place in this layer. The output of the model or inferencing results will also be stored in the Product Datalake.

The core of this stack lies in the ability for Azure Machine Learning to be able to access and interface with the Datalake Store. The rest of the paper will focus on how this can be achieved.

Azure Machine Learning

Azure Machine Learning Service [2] (AMLS) provides a cloud-based environment for the development, management and deployment of machine learning models. The service allows the user to start training models on local machine and then scale out to the cloud. The service fully supports open-source technologies such as PyTorch, TensorFlow, and scikit-learn and can be used for any kind of machine learning, from classical ml to deep learning, supervised and unsupervised learning.

Figure 3 below shows the architectural pattern using that focuses around product Datalake for interacting with the Azure ML. Until recently, the data used for model training needed to either reside in the default (blob or file) storage associated with the Azure Machine Learning Service Workspace, or a complex pipeline needed to be built to move the data to the compute target during training. This has meant that data stored in Azure Data Lake Store Gen1 (ADLSG1) typically needed to be duplicated to the default (blob of file) storage before training could take place. This is no longer necessary with the new feature Dataset.

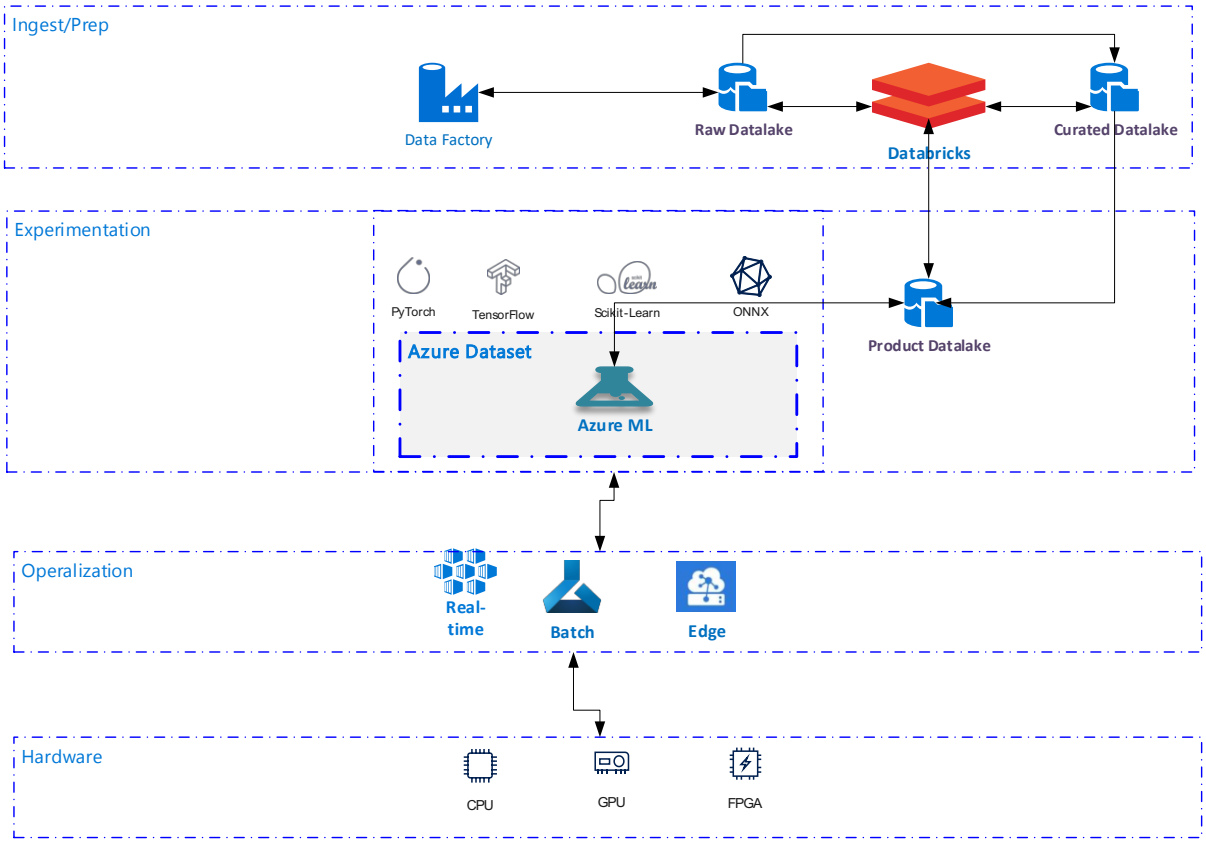

Fig 3 - ADLS Architectural Pattern for MLS

Azure Dataset

The new Dataset feature of AMLS has made it possible to work with data files stored in ADLS-G1 by creating a reference to the data source location, along with a copy of its metadata. The data remains in its existing location, so no extra storage cost is incurred. The data set is thus identified by name and is made available to all workspace users to work with.

Simple Example

In the following we will demonstrate how we can use the Azure Datasets with Azure Maching learning to build a machine learning model using the Product Datalake, the steps are as follows:

- Step 1: Register the Product Datalake (ADLS Gen1) store as a data store in the AMLS workspace
- Step 2: Register a file (CSV) as a dataset in the AMLS workspace
- Step 3: Train a Model

Step 1: Register the Product Datalake as a data store in the AMLS workspace

We will be using Azure ML Notebook VM to implement to demonstrate this example [3], this because it comes prebuilt with the Python Jupyter Notebook with MLS sdk installed. However, if you prefer to use your own IDE, you will need to install the MLS python SDK.

```
from azureml.core.workspace import Workspace
from azureml.core.datastore import Datastore
from azureml.core.dataset import Dataset
from azureml.data.data_reference import DataReference

#Give a name to the registered datastore
datastore_name = "adlsdatastorex"

#Get a reference to the AMLS workspace
workspace = Workspace.from_config()

#Register the Data store pointing at the ADLS G1
Datastore.register_azure_data_lake(workspace,
                    datastore_name,
                    "<Name of the Product Datalake>",
                    "<AAD Tenant ID>",
                    "<Service Principal Application ID>",
                    "<Service Principal Secret>",
                    overwrite=False)
```

The above code snippet assumes that the AMLS configuration file is available in the working directory. The file can be downloaded from the Azure portal by selecting “Download config.json” from the Overview section of the AMLS workspace. Or you can create it yourself:

```
{
    "subscription_id": "my subscription id",
    "resource_group": "my resource group",
    "workspace_name": "my workspace name"
}
```

We also need to register an application (Service Principal) with the Azure Active Directory (AAD) tenant with Read access on the data lake store files we need to use.

Note: You should use a Key Vault to store the Service principal ID & Secrets.

At this stage the ADLSG1 should be registered as a datastore in the workspace. This can be tested using the following, which should return an azureml.data.azure_data_lake_datastore.AzureDataLakeDatastore object:

```
# retrieve an existing datastore in the workspace by name
dstore = Datastore.get(workspace, datastore_name)
print(dstore)
```

Step 2: Register a CSV file as a dataset in the AMLS workspace:

```
filepath='file1.csv'
dset = Dataset.from_delimited_files(DataReference(dstore, path_on_datastore=filepath, mode='mount'))
```

Now the data file is available as a data frame in memory, for example to return the first 5 rows:

```
dset.head(5)
```

To register the dataset with the workspace:

```
dset = dset.register(workspace, 'myDataSet')
```

To get a list of the datasets registered with the workspace:

```
print(Dataset.list(workspace))
```

Step 3: Train a Model

```
#Get the dataset that is already registered with the workspace
data_set =Dataset.get(workspace,'CricketChirps')

#Use the dataset
```

```
dataset=data_set.to_pandas_dataframe()
```

The above code will get an existing registered dataset, and covert it to a Pandas dataframe. You can then conduct additional pre-processing and feature engineering. You also have the option of reading it as a Spark dataframe.

```
X = dataset.iloc[:, :-1].values  #  independent variable
y = dataset.iloc[:, 1].values    #  dependent variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
run = Run.get_context()
print('Train a linear regression model')
regressor = LinearRegression()    # This object is the regressor, that does the regression
regressor.fit(X_train, y_train)   # Provide training data so the machine can learn to predict using a learned model.

y_pred = regressor.predict(X_test)
print(y_pred)
```

The above code splits the dataset into training and testing as well as input and output features. A simple linear regression model is trained and validated using the validation dataset.

For full source code, please visit: https://github.com/mufajjul/mls-adls-dataset

References

[1] https://azure.microsoft.com/en-gb/services/storage/data-lake-storage/

[2] https://azure.microsoft.com/en-gb/services/machine-learning-service/

[3] https://docs.microsoft.com/en-us/azure/machine-learning/service/tutorial-1st-experiment-sdk-setup