



Building the Data Lake

with Microsoft Azure Data Factory and Data Lake Analytics

Khalid M. Salama, Ph.D.
Business Insights & Analytics
Hitachi Consulting UK

Outline

- What is a Data Lake?
- Data Lake Architecture & Technologies
- Data Ingestion – Populating the Data Lake
- Azure Data Lake Store Overview
- Introducing Azure Data Factory
- Data Lake Ingestion Scenarios with Azure Data Factory
- Introduction to Azure Data Lake Analytics
- Processing Data Using U-SQL + .net Extensibility
- Running U-SQL Jobs on Azure

Data Lake – The Concept

What is that? And Why it is needed?

What is a Data lake

A repository that holds raw data file extracts of all the Enterprise source systems

Usually a Distributed File System (HDFS, Blob Storage, Amazon S3, GFS, etc.) to support Big Data Processing

Pentaho (Hitachi Data Systems) co-founder and CTO, James Dixon coined the term ‘Data Lake’

Why a Data Lake

Single and cheap repository of Enterprise-wide (raw) data

Abstracts your DW/ Business Intelligence Solutions from source systems

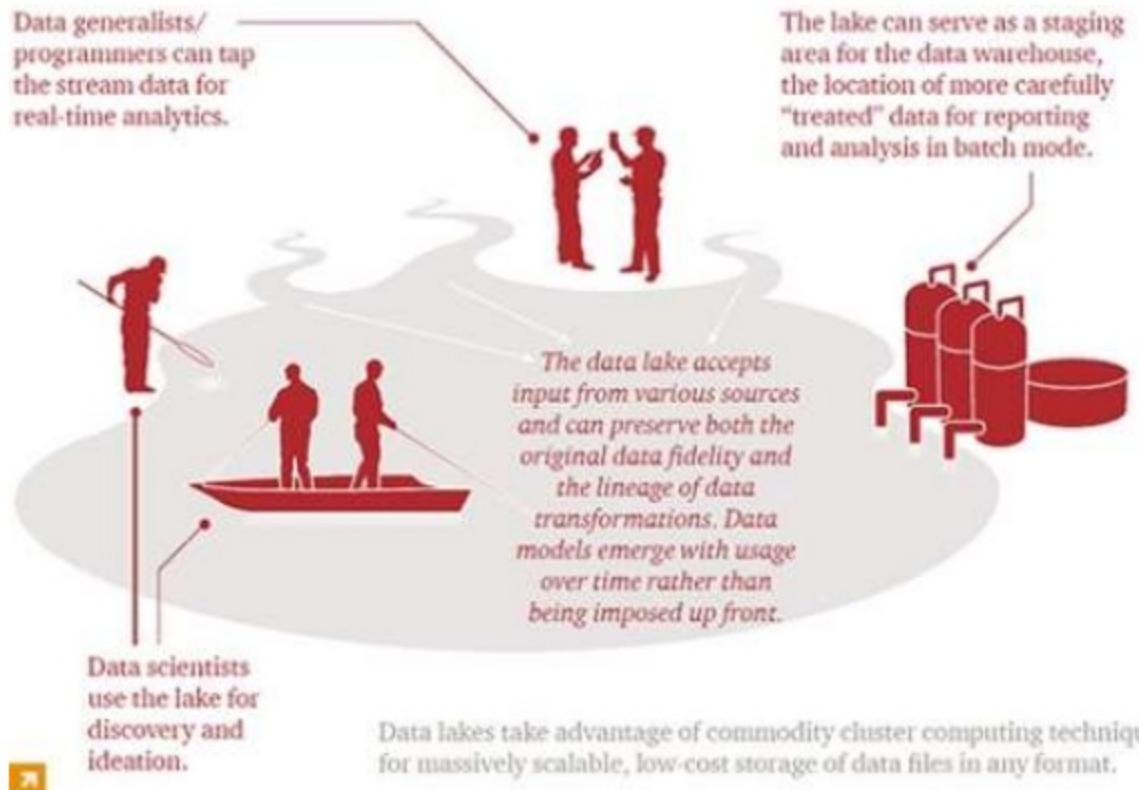
Supports potential Big Data Processing and Analytics

Data Lake – The Concept

What is that? And Why it is needed?

*If you think of a datamart/DW as a store of **bottled water** – cleansed, packaged and structured for easy consumption – the Data lake is a **large body of water** in a more natural state.*

The contents of the data lake, stream in from many source to fill the lake, and various users of the lake can come to examine, dive in, or take samples.”



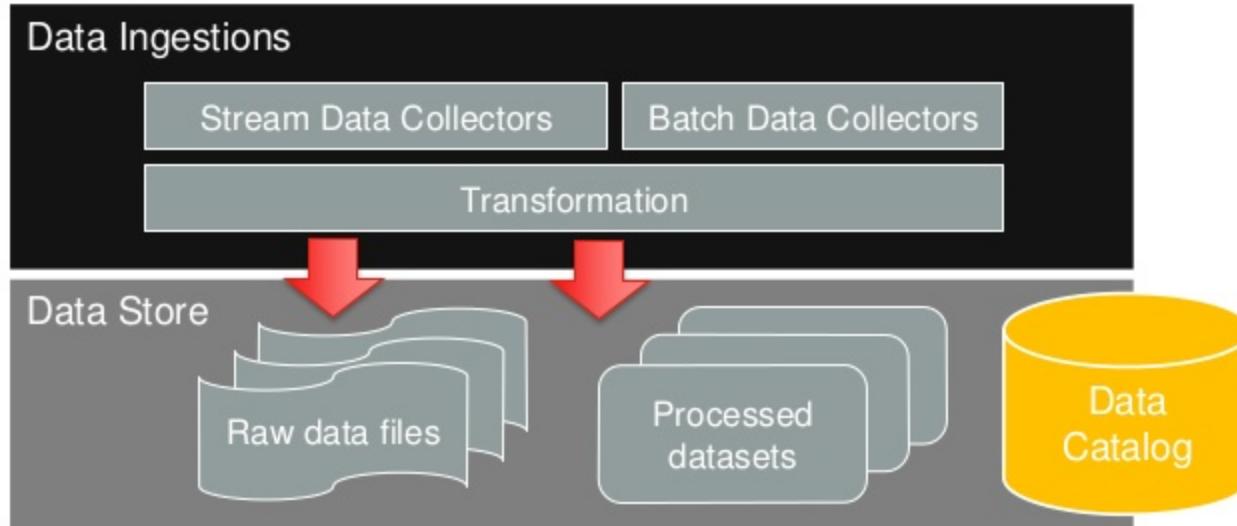
Data Lake

Overall Architecture



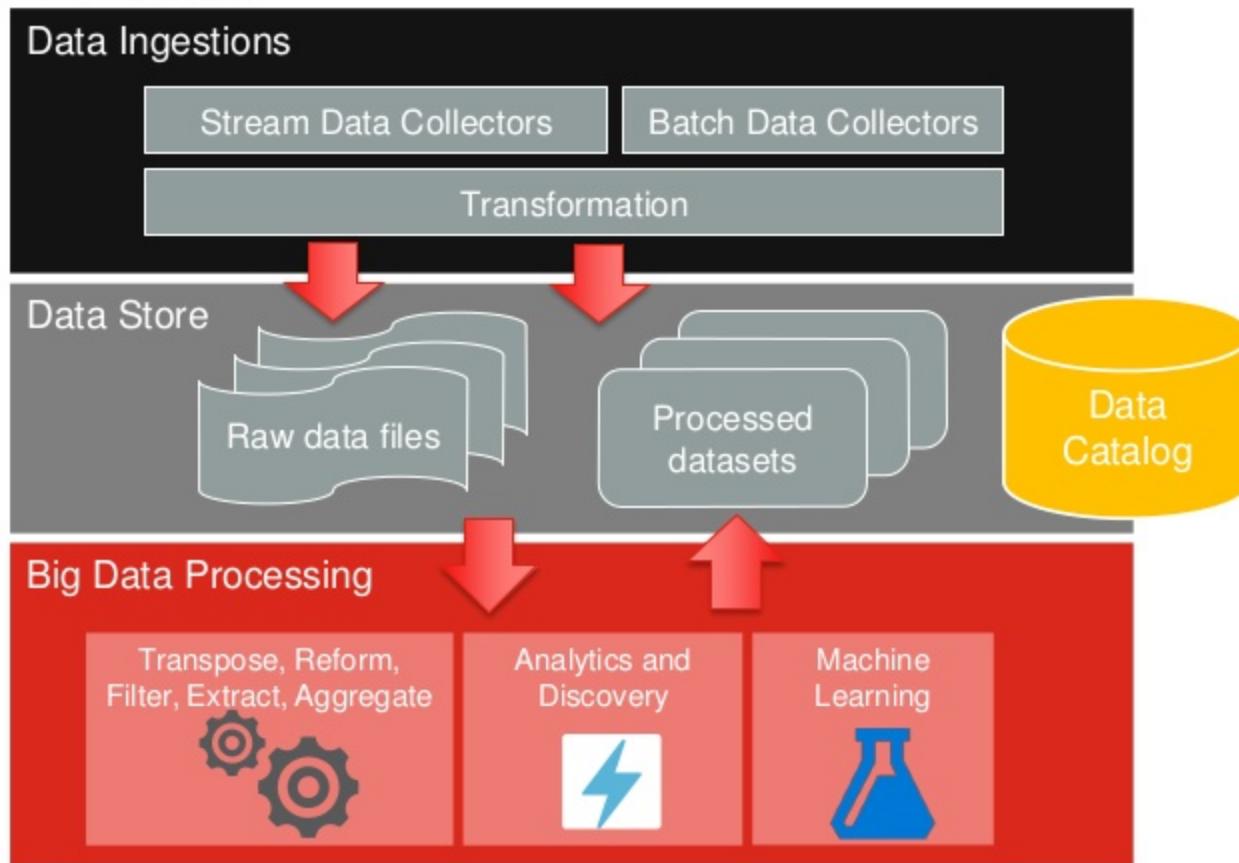
Data Lake

Overall Architecture



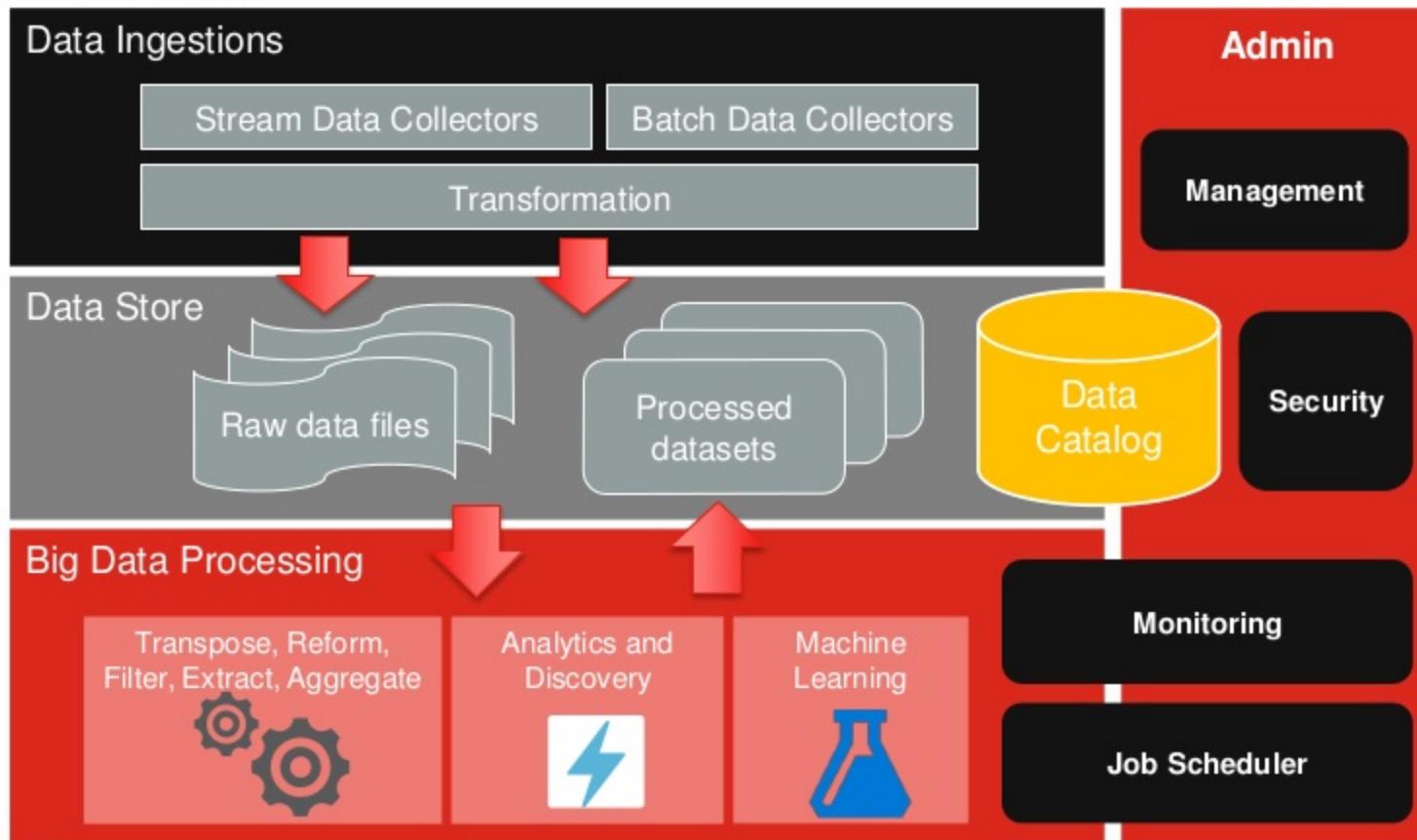
Data Lake

Overall Architecture



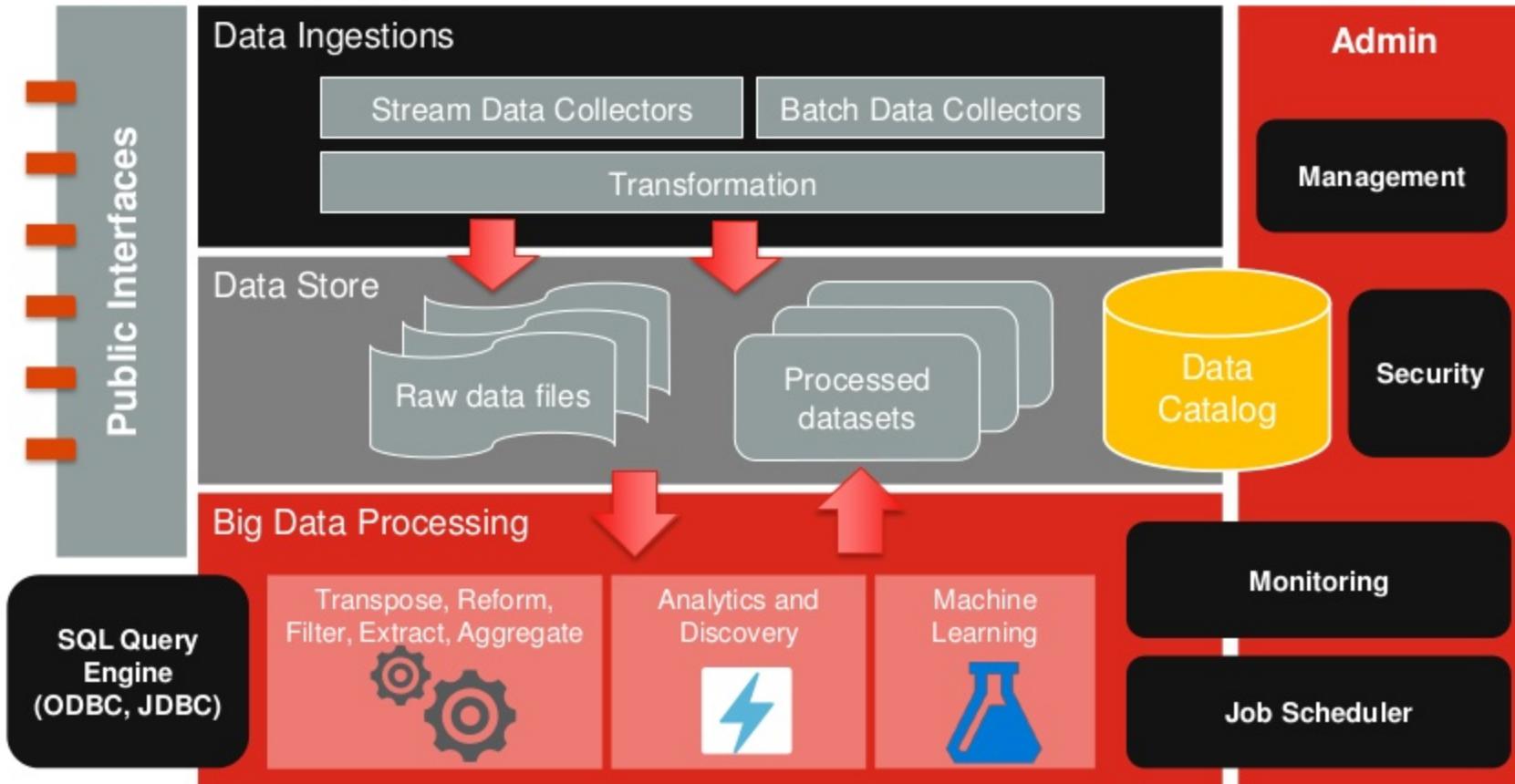
Data Lake

Overall Architecture



Data Lake

Overall Architecture



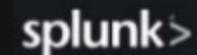
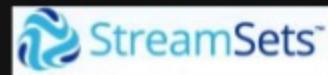
Data Lake

Overall Architecture

Data Ingestion	Batch Data Collectors	Pull data from data sources using batch connectors E.g. SQL, NoSQL, File Systems, (s)FTP, etc.
	Stream Data Collectors	Consumes data streams in using real-time data connectors. E.g. queues (kafka, rabbitmq, MQTT, eventshub, etc.), WebSockets, http, etc.
	Transformation	Performs basic data transformation & enrichment, rather than processing, including format conversion (encoding), compression, merge, split, etc.
Data Store (Distributed File System)	Raw Data Files	Data stored in files in its raw form (either in its original format, or a standardized format/compression applied in the transformation stage) – XML, JSON, BSON, Text, Avro, Parquet, ORC, etc.
	Processed Datasets	The results of (big) data processing jobs, which is a set of tabular dataset (files) in a query-optimized format
	Data Catalog	Usually a relational database that includes a catalog for the datasets in the data lake
Big Data Processing	Data Processing Jobs	Big data processing jobs (MapReduce, Hive, Pig, Spark, U-SQL) that process the data and apply advanced analytics, including ML and pattern discovery
	Jobs Scheduler	A scheduler that Orchestrate the execution of the data processing jobs
SQL Query Engine		ODBC, JDBC APIs to query the datasets in the data lake using SQL-like scripts
Public APIs		APIs for retrieving the datasets in the data lake - In some cases, for ingesting data

Data Lake Technologies

Data Ingestion



Data Store

(Distributed File System)

HDFS, GFS, Azure Blob Storage, Azure Data Lake, S3, etc.

Jobs Scheduler

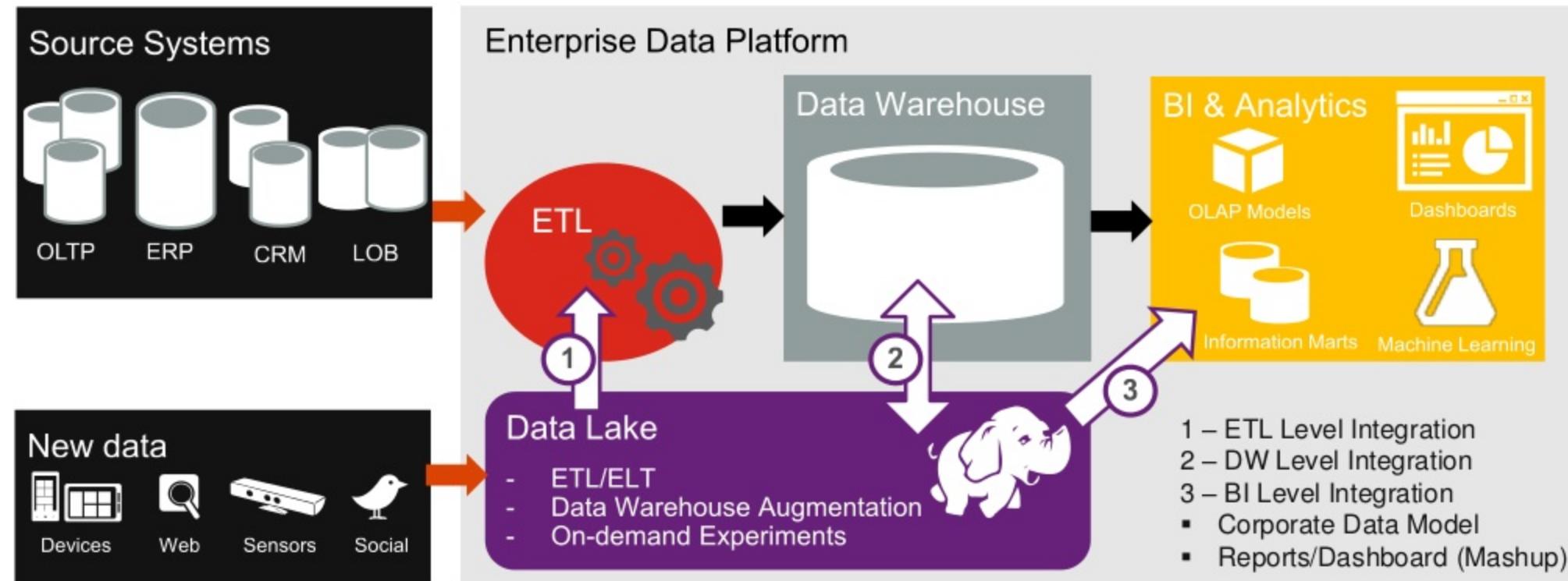


Data Processing

MapReduce



Data Lake & Enterprise Data Platform

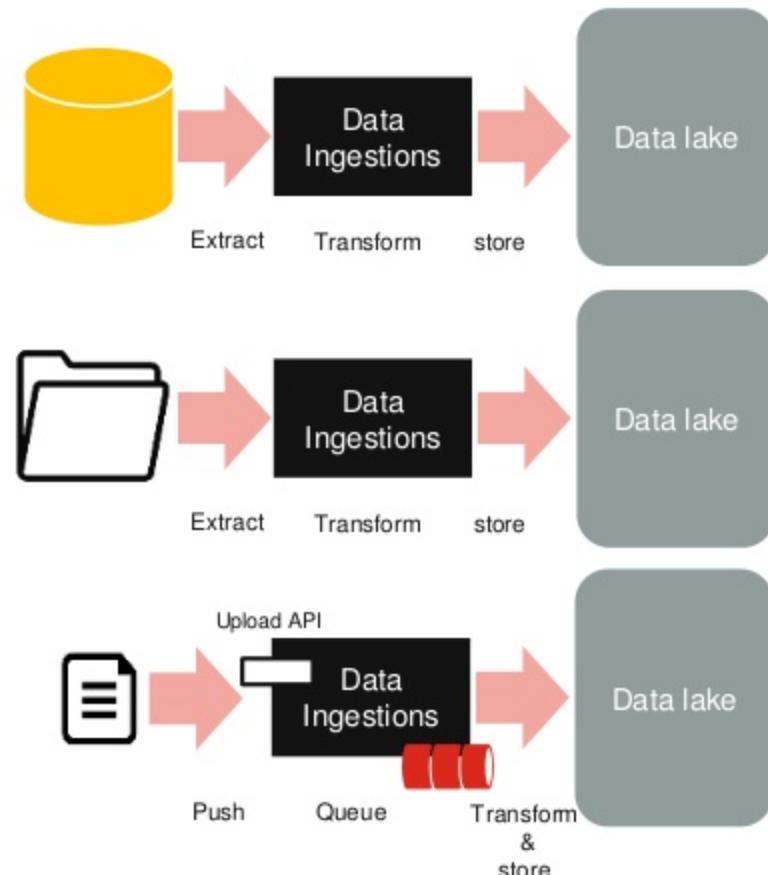


Data Ingestion – Populating the Data Lake

Data Ingestion

Batch Data Ingestion

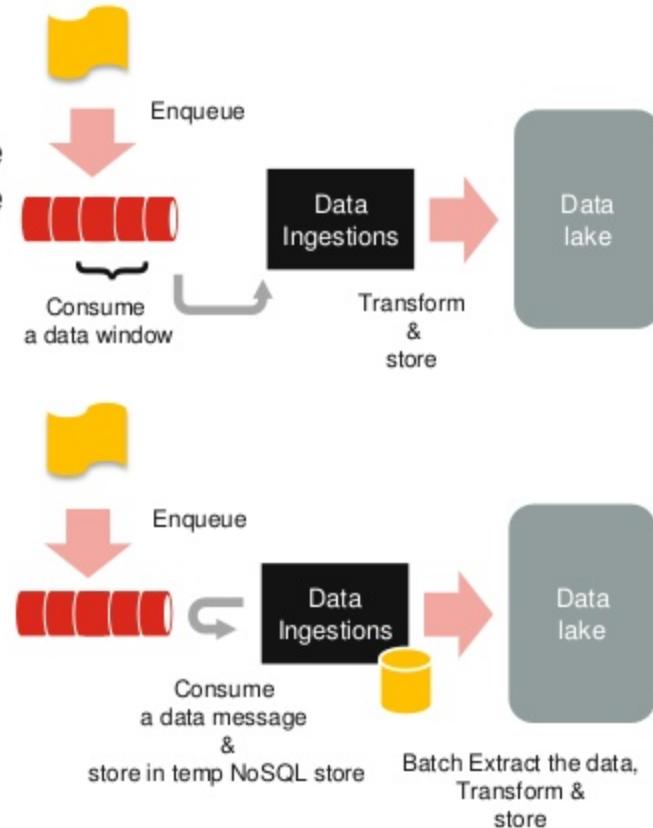
- Refers to ingesting **large amount** of data into the data lake in **low-frequency** (hourly, daily, monthly, etc.)
- Data sources
 - RDBMS
 - NoSQL data stores
 - File systems
 - Blob storages
- Data is extracted from the source using time **window filter** (from-to), or based on **existing files** in the source location.
- If data file is pushed to data lake using **ingestion APIs**, they are received in **an internal queue** before they are consumed and stored in data lake
- Runs on **schedule**



Data Ingestion

Real-time Data Ingestion

- Refers to ingesting **small data messages** into the data lake in **high-frequency** (near-real time)
- In some scenarios, data message are pushed into a **message queue**, where the data ingestion tool **consumes** these messages from the queue and store them in the data lake.
- In other scenarios, the data generator calls an **ingestion API** to push the data to the data lake, where the **data message is received initially in an internal queue** before it is transformed and stored in the lake
- If the data **message is small**, it is preferable to consume a “**window**” of **messages** at a time before the are **stored (as one file)** in the data lake to avoid having many small files, one per each message in the data lake.
- One idea is to store each message in a **intermediate NoSQL store**, in real time, where a batch data ingestion retrieve all the data inserted at once to store them in one file in the data lake



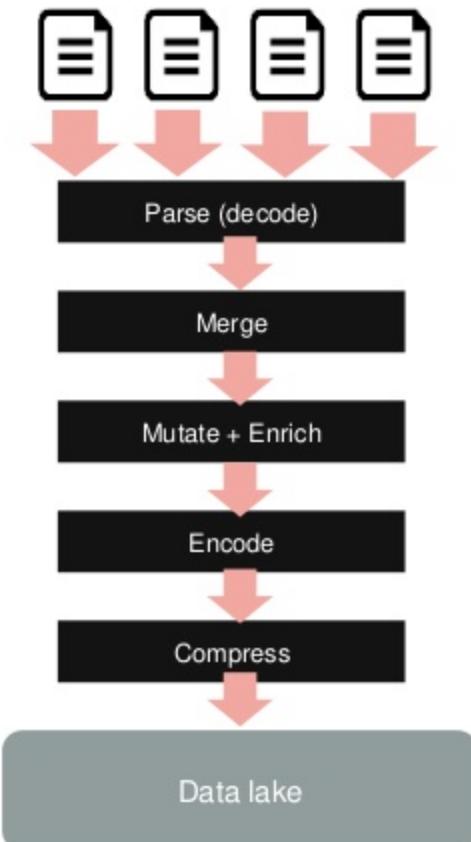
Data Ingestion

Transformation

Data ingestion tools are **not meant to perform data processing**, rather some data **transformation** tasks are applied during ingesting data to data lake:

- **File format change** (decode/encode) – optimize the data file for read/write
- **File compression** before store – optimize storage and files shuffling
- **Merging** multiple file (or data messages) from source into one file to be stored (to avoid having many small files)
- Adding **metadata** columns to the dataset in-flight (enrichment)
- **Mutate**: Removing, renaming columns, etc.

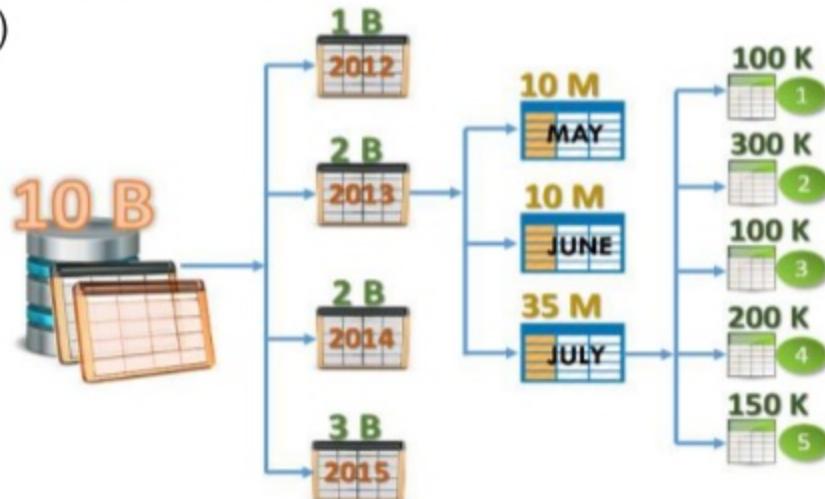
In some cases, if the source data is a file system, **binary (blind) file ingestion** can be performed (without parsing the file content) if no transformations or encoding are needed – maybe just compression



Data Ingestion

Data Partitioning

- The incoming data is partitioned and the files are stored in different **(hierarchical) directories** in the file system
- Partitioning is typically based on date/time (year/month/day/...) and/or any data element used in the “where” clause (country, site, category, etc.)
- The reason is achieve **partitions elimination** while processing data (if the partition is part of the **where/filter criteria** of the processing)
- Improves the performance** of the query/processing jobs as **less data files** are considered.



Data Ingestion

File Formats & Compression

Raw Files

- Raw data files maybe stored in its **original format** (TEXT/XML/JSON)
- Original format **allow users** to download the original files and **work with them**
- **Compression** is applied to improve **query/processing performance**
 - For very **large** file, **splittable** compression is applied (e.g. bzip2, LZO) – usually **slow to compress**
 - For **normal** files, high speed (**less efficient**) compression can be applied (e.g. gzip, deflate)
- **Splitability** is important as multiple **processing** task can work on different parts of the file **simultaneously**

Processes Data Files

- **All** the processed data files (dataset) should follow the **same format**
- Since the processed datasets are query intensive (used to perform aggregations, etc.), **Columnar-oriented formats** are recommended (for **better read performance**) – e.g. Parquet, ORC
- For supporting **schema evolution** (e.g. expected changes in schema), formats Parquet and Avro are recommended.
- Since Columnar-oriented format are binary and compressed, no compression, or snappy (fast, less effecient) compression can be applied

Data Ingestion

File Formats

Format	Description	Content	Splitable	Schema Evolution	Size	Read Perform.
TEXTFILE	<ul style="list-style-type: none"> ▪ Delimited text file ▪ Any Platform 	Plain Text	Yes	No	6	6
SEQUENCEFILE	<ul style="list-style-type: none"> ▪ Binary key-value pairs ▪ Optimized for MapReduce 	Plain Text	Yes	No	5	4
PARQUET	<ul style="list-style-type: none"> ▪ Columnar storage format ▪ Optimized for querying "wide" tables 	Binary (Compressed)	No	Yes	2	2
RCFile	<ul style="list-style-type: none"> ▪ Columnar storage format 	Binary (Compressed)	No	No	3	3
ORC	<ul style="list-style-type: none"> ▪ Columnar storage format ▪ Optimized for distinct selection and aggregations ▪ Vectorization: Batch Processing. Each batch is a column vector. 	Binary (Highly Compressed)	No	No	1	1
AVRO	<ul style="list-style-type: none"> ▪ serialization system with evolvable schema-driven binary data ▪ Cross-platform inter-operability. 	Binary	Yes	Yes	4	5

Data Ingestion

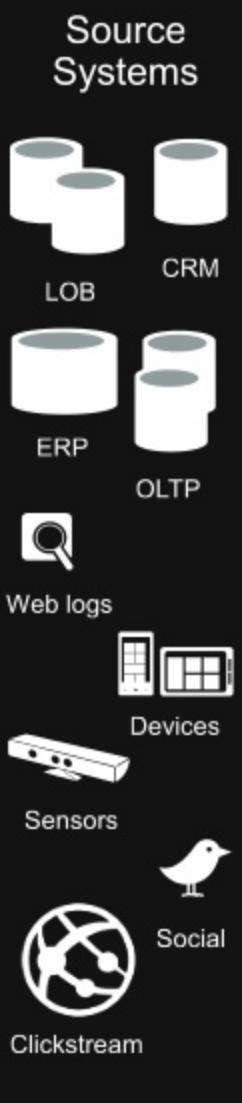
Compression

Compression	Extension	Codec	Splittable	Efficacy (Compression Ratio)	Speed (to compress)
DEFLATE	.deflate	org.apache.hadoop.io.compress.DefaultCodec	No	Medium	Medium
gzip	.gz	org.apache.hadoop.io.compress.GzipCodec	No	Medium	Medium
bzip2	.bz2	org.apache.hadoop.io.compress.BZip2Codec	Yes	High	Low
LZO	.lzo	com.hadoop.compression.lzo.LzopCodec	Yes*	Low	High
LZ4	.lz4	org.apache.hadoop.io.compress.Lz4Codec	No	Low	High
Snappy	.snappy	org.apache.hadoop.io.compress.SnappyCodec	No	Low	High

Reduces space needed to store files

Speeds up data transfer across the network or to or from disk

Azure Data Lake Store



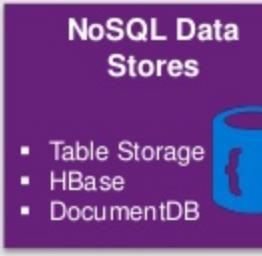
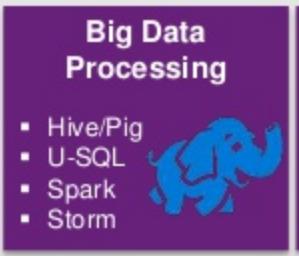
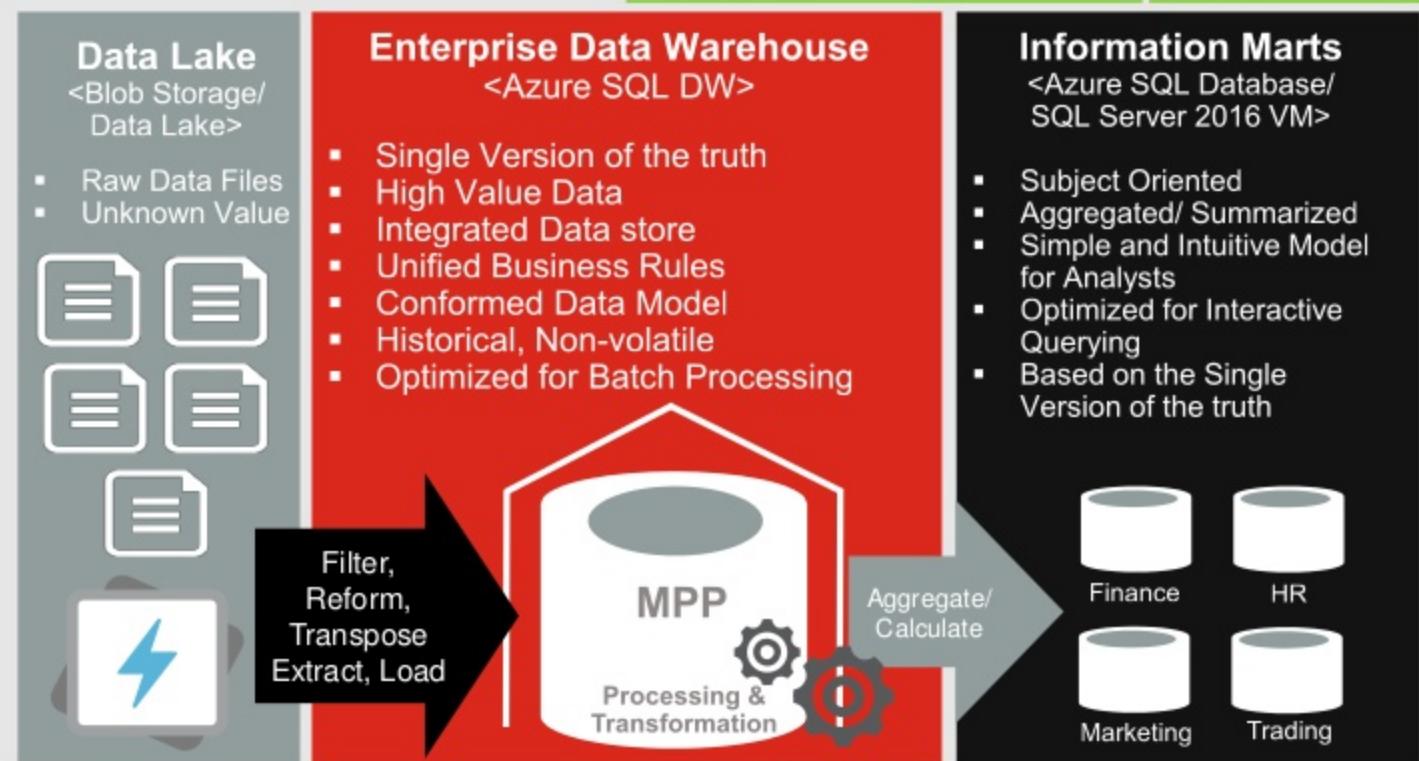
Azure Enterprise Data Platform

High Performance Computing

- Azure Batch

Data Orchestration & Movement

- Azure Data Factory



Azure Data Lake Store

Azure Data Lake Services

HDInsight

- Hadoop Cluster as a Service
- MapReduce | Pig | Hive | Spark
- HBase | Oozie | Storm
- Unit of Cost: Running nodes



Data Lake Analytics

- Big Data Jobs as a Service
- Built on Apache YARN
- U-SQL
- Unit of Cost: running Job



Data Lake Store

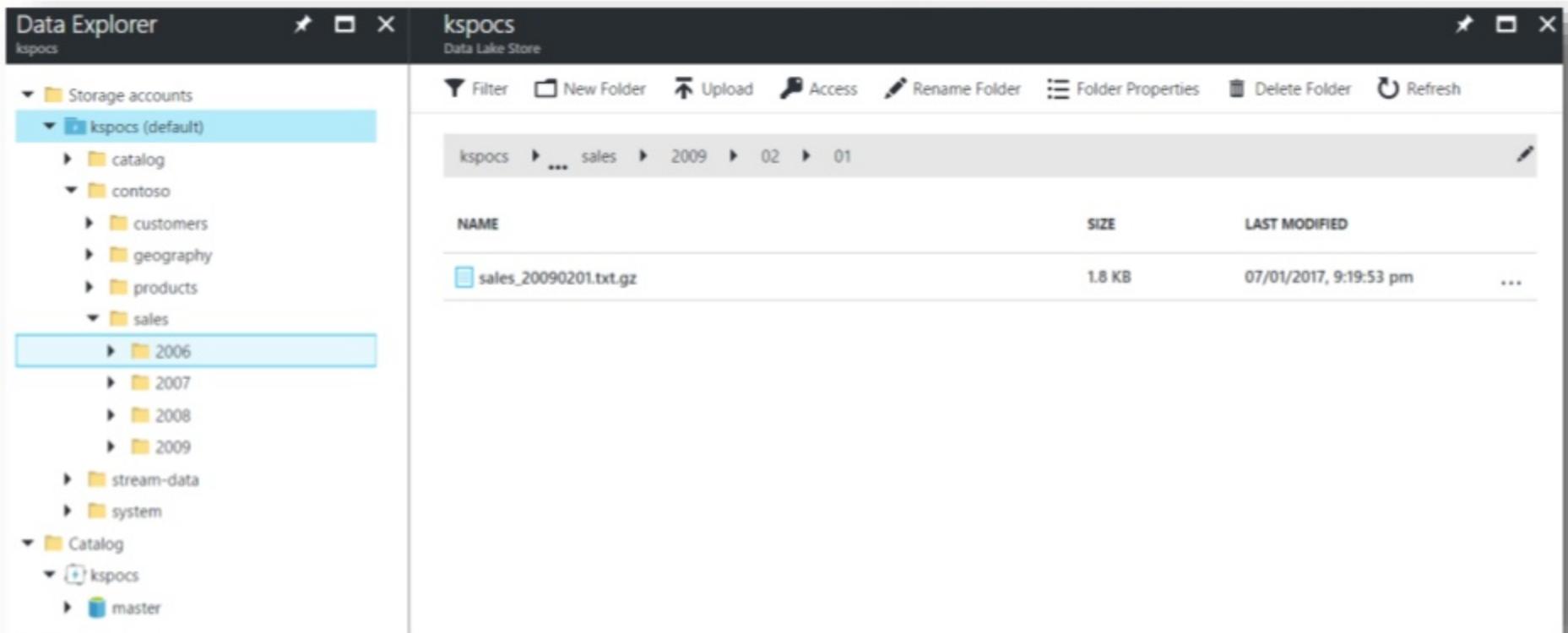
- Built for data files, rather than Blobs
- WebHDFS REST APIs for Hadoop integration
- Optimized for distributed analytical workloads
- Unlimited Storage, petabyte files
- Security: AAD and ACL, Encryption
- Built-in Catalog for “virtual” databases and .net assemblies



Azure Data Lake Store

Azure Portal

adl://<data_lake_store_name>.azuredatalakestore.net



The screenshot shows the Azure Data Explorer interface for the Data Lake Store 'kspocs'. The left pane displays a hierarchical file system:

- Storage accounts
- kspocs (default)** (selected)
 - catalog
 - contoso
 - customers
 - geography
 - products
 - sales** (selected)
 - 2006
 - 2007
 - 2008
 - 2009
 - stream-data
 - system- Catalog
- kspocs** (selected)
- master

The right pane shows the contents of the 'sales' folder in the '2009' subdirectory:

NAME	SIZE	LAST MODIFIED	...
sales_20090201.txt.gz	1.8 KB	07/01/2017, 9:19:53 pm	...

Azure Data Lake Store

Ingest Data

Interactively

- Azure Portal
- Azure PowerShell
- Azure Cross-platform CLI
- Data Lake Tools for Visual Studio
- AdlCopy Tool: From Blob



Data Lake Store



Azure Stream Analytics

- Real-time
- Event-based
- From Events Hub/IoT Hub



Programmatically

- .Net SDK
- Java SDK
- Node.js SDK
- REST APIs



Azure Data Factory

- Batch
- Scheduled
- From SQL/NoSQL/File Systems



Azure HDInsight

- Sqoop: From SQL
- Distcp: From Blob/HDFS
- Oozie is used for Scheduling

Azure Data Lake Store

Secure Data



Authentication

- Azure Active Directory (ADD)
- Users and service identities defined in your ADD can access your Data Lake Store account,
- Key advantages of using Azure Active Directory as a centralized access control mechanism are:
 - Simplified identity lifecycle management.
 - Authentication from any client through a standard open protocol, such as OAuth or OpenID.
 - Federation with enterprise directory services and cloud identity provider

Authorization

- Role-based access control (Owner/Reader/Contributor/Administrator)
- Access Control list (Read/Write/Execute)
- Files and folders both have Access ACLs

Data Protection

- In transit - Transport Layer Security (TLS) protocol to secure data over the network.
- At rest - provides transparent encryption for data that is stored in the account.
 - Data Lake Store automatically encrypts data prior to persisting and decrypts data prior to retrieval,
 - Completely transparent to the client accessing the data

Network Isolation

- Firewall with IP address range for your trusted clients



Azure Data Lake Store

.Net SDK

NuGet Package Manager: ADLStoreTest

Package source: nuget.org

Microsoft.Azure.Management.DataLake.

Version: Latest stable 1.0.3 Install

Options

Description

Provides Data Lake Store account and filesystem management capabilities for Microsoft Azure.

Version: 1.0.3 Author(s): Microsoft

```
var adlClient = new DataLakeStoreFileSystemManagementClient(credentials);
adlClient.FileSystem.
```

- Append
- AppendAsync
- AppendWithHttpMessagesAsync
- CheckAccess
- CheckAccessAsync
- CheckAccessWithHttpMessagesAsync
- Concat
- ConcatAsync
- ConcatWithHttpMessagesAsync

(extension) void IFileSyste
[Microsoft.Azure.Manageme

using Microsoft.Azure.Management.DataLake.Store;

Azure Data Lake Store

Data Lake vs Blob Storage

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

	Azure Data Lake Store	Azure Blob Storage
Purpose	Optimized performance for parallel analytics workloads. High Throughput and IOPS.	General purpose object store for a wide variety of storage scenarios
Use Cases	Batch, interactive, streaming analytics and machine learning data such as log files, IoT data, click streams, large datasets	Any type of text or binary data, such as application back end, backup data, media storage for streaming and general purpose data
Key Concepts	Data Lake Store account contains folders, which in turn contains data stored as files	Storage account has containers, which in turn has data in the form of blobs
Structure	Hierarchical file system	Object store with flat namespace
HDFS Client	Yes	Yes
Server-side API	WebHDFS-compatible REST API	Azure Blob Storage REST API
Security - Authentication	Based on Azure Active Directory Identities	Based on shared secrets - Account Access Keys and Shared Access Signature Keys.
Security - Authorization	POSIX Access Control Lists (ACLs). ACLs based on Azure Active Directory Identities can be set file and folder level.	For account-level authorization – Use Account Access Keys For account, container, or blob authorization - Use Shared Access Signature Keys
Encryption data at rest	Transparent, Server side	Transparent, Server side
Developer SDKs	.NET, Java, Python, Node.js	.Net, Java, Python, Node.js, C++, Ruby
Size limits	No limits on account sizes, file sizes or number of files	Specific limits documented here
Redundancy	Locally-redundant (multiple copies of data in one Azure region)	Locally redundant (LRS), globally redundant (GRS), read-access globally redundant (RA-GRS). See here for more information

Azure Data Factory

Azure Data Factory

Cloud data processing & movement services

A managed Azure cloud service for building & operating data pipelines

Scalable

Reliable

Pay-as-you use

Batch Data Ingestion & Movement

No real-time data collection (use stream Analytics instead)

Format conversion, compression, files merge, column mapping

Data Processing Orchestration and Scheduling

Not a data processing engine, but uses different compute platforms



Azure Data Factory

Concepts & Components

Data Factory



Linked Service



A connection information to a data store or a compute resource

Dataset



A pointer to the data object to be used as input or an output of an Activity

Pipeline



logical grouping of Activities with certain sequence & schedule

Azure Data Factory

Concepts & Components

Data Factory



Pipeline



Activity



actions to perform on data.

Input 0-N dataset(s) - Output 1-N dataset(s)

Linked Service



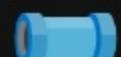
A connection information to a data store or a compute resource

Dataset



A pointer to the data object to be used as input or an output of an Activity

Pipeline



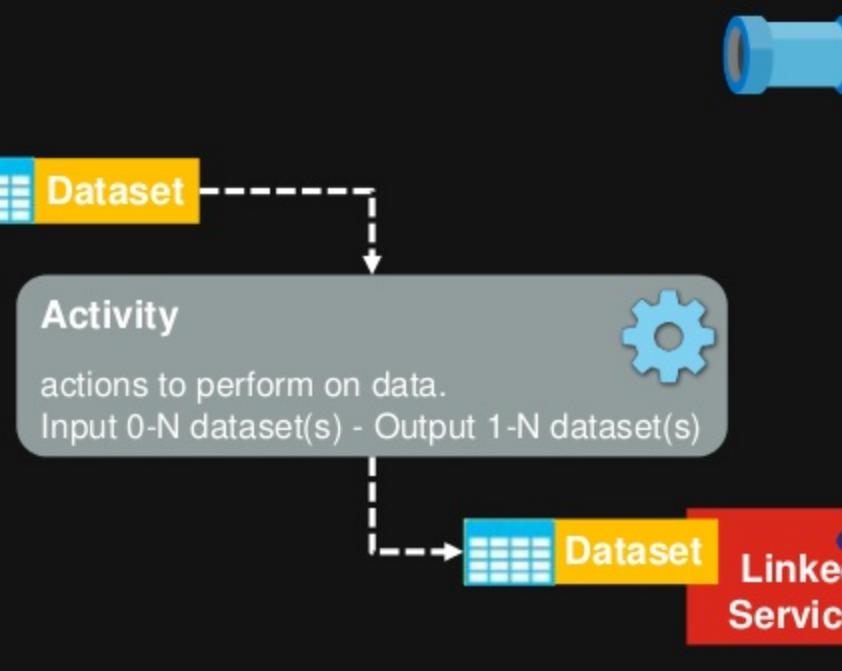
logical grouping of Activities with certain sequence & schedule

Azure Data Factory

Concepts & Components

Data Factory

Pipeline



Linked Service



A connection information to a data store or a compute resource



Dataset

A pointer to the data object to be used as input or an output of an Activity



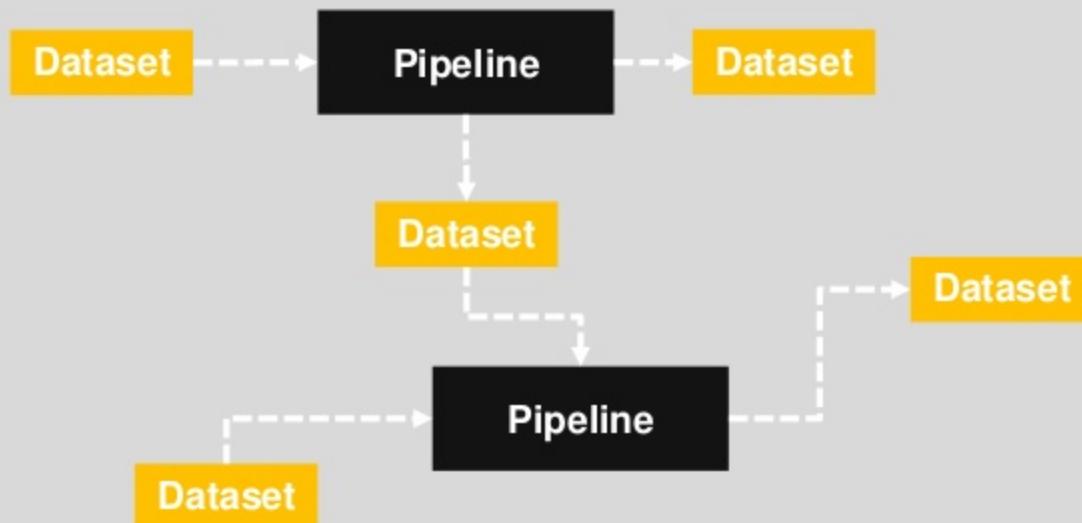
Pipeline

logical grouping of Activities with certain sequence & schedule

Azure Data Factory

Concepts & Components

Data Factory



Linked Service



A connection information to a data store or a compute resource

Dataset



A pointer to the data object to be used as input or an output of an Activity

Pipeline



logical grouping of Activities with certain sequence & schedule

Azure Data Factory

Concepts & Components



Azure Data Factory

Features

Linked Service – Data

- Azure Blob
- Azure Table
- Azure SQL Database
- Azure SQL Data Warehouse
- Azure DocumentDB
- Azure Data Lake Store
- SQL Server
- ODBC data sources
- Hadoop Distributed File System (HDFS)

Linked Service - Compute

- Azure HDInsight On-Demand
- Azure Batch
- Azure Machine Learning
- Azure Data Lake Analytics
- Azure SQL DW (StoredProcedure)
- Azure SQL DB (StoredProcedure)
- SQL Server (StoredProcedure)

Activities

- Data Movement
(copy Data from Source to sink)
- Data Transformation
 - (on-demand) HDInsight
(Hive | Pig | MapReduce | Spark)
 - Azure ML Batch
 - Stored Procedure
 - Data Lake U-SQL Jobs
 - Custom .NET
(Azure Batch)

Data Gateway - Connect to an on-prem data source

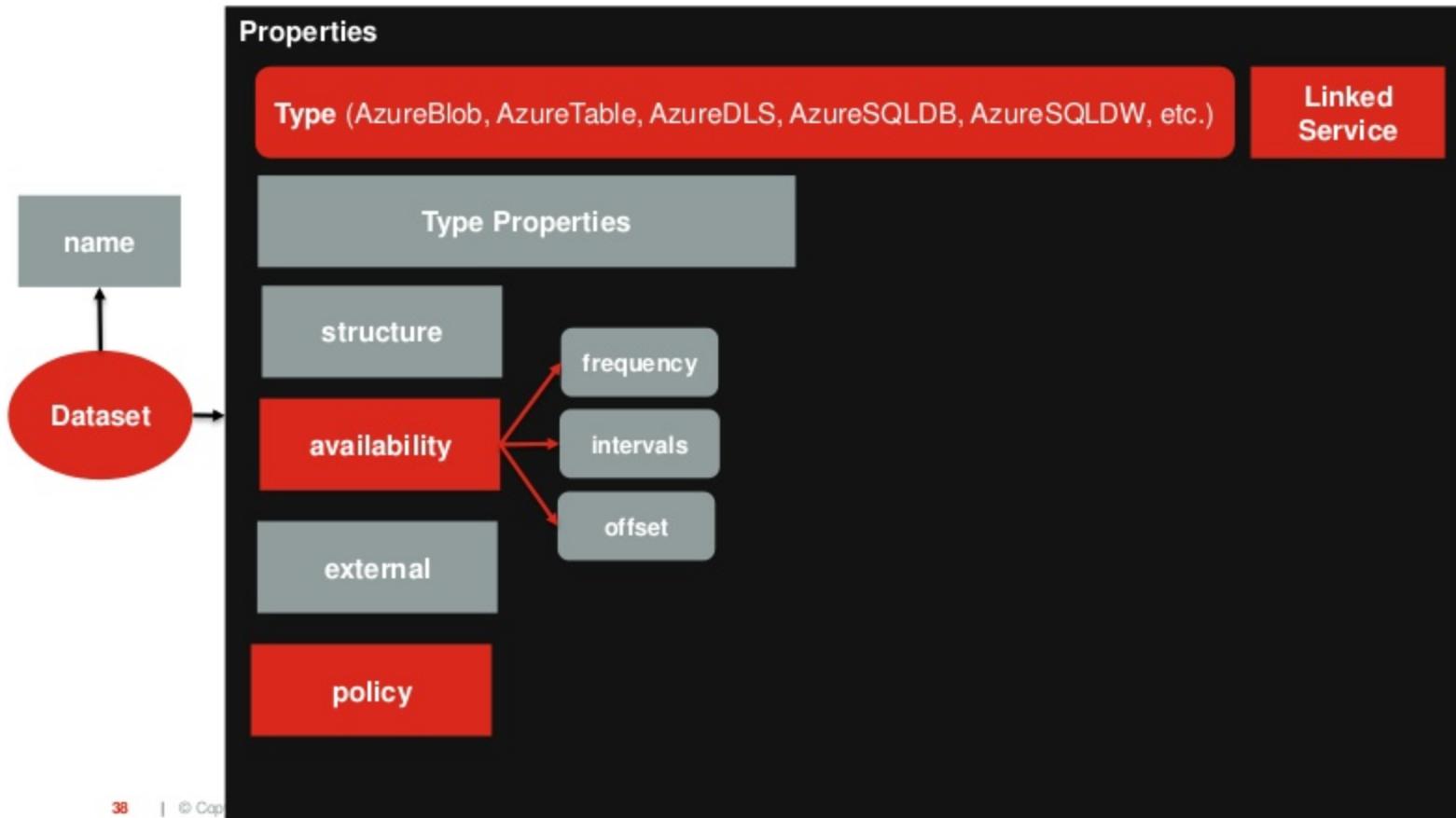
Azure Data Factory - Taxonomy

Dataset

```
{  
    "name": "input-azblobstorage-test_data_v8",  
    "properties": {  
        "structure": [...],  
        "published": false,  
        "description": "true",  
        "type": "AzureBlob",  
        "linkedServiceName": "azstorage-kspocs",  
        "typeProperties": [...],  
        "availability": [...],  
        "external": true,  
        "policy": [...]  
    }  
}
```

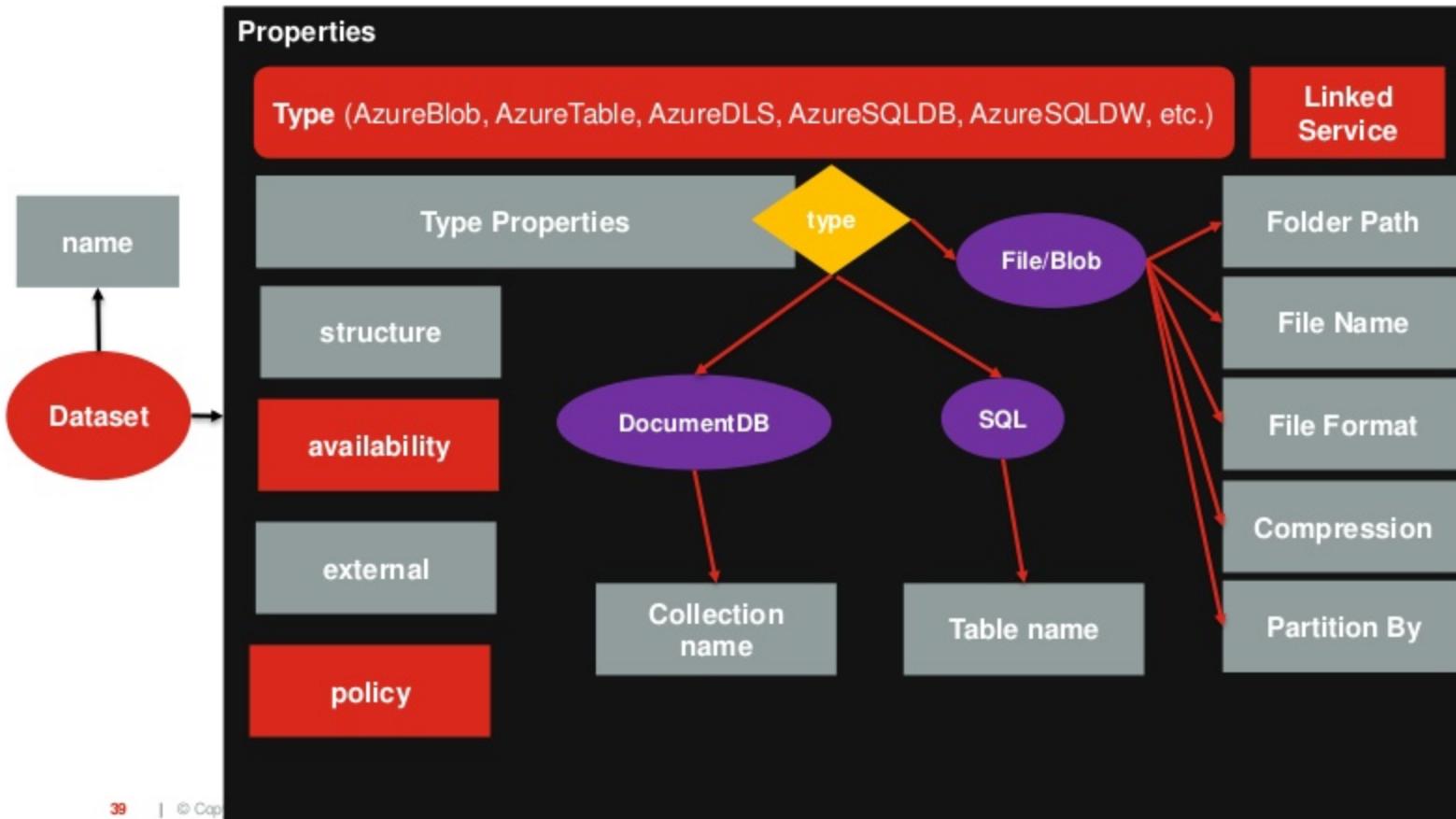
Azure Data Factory - Taxonomy

Dataset



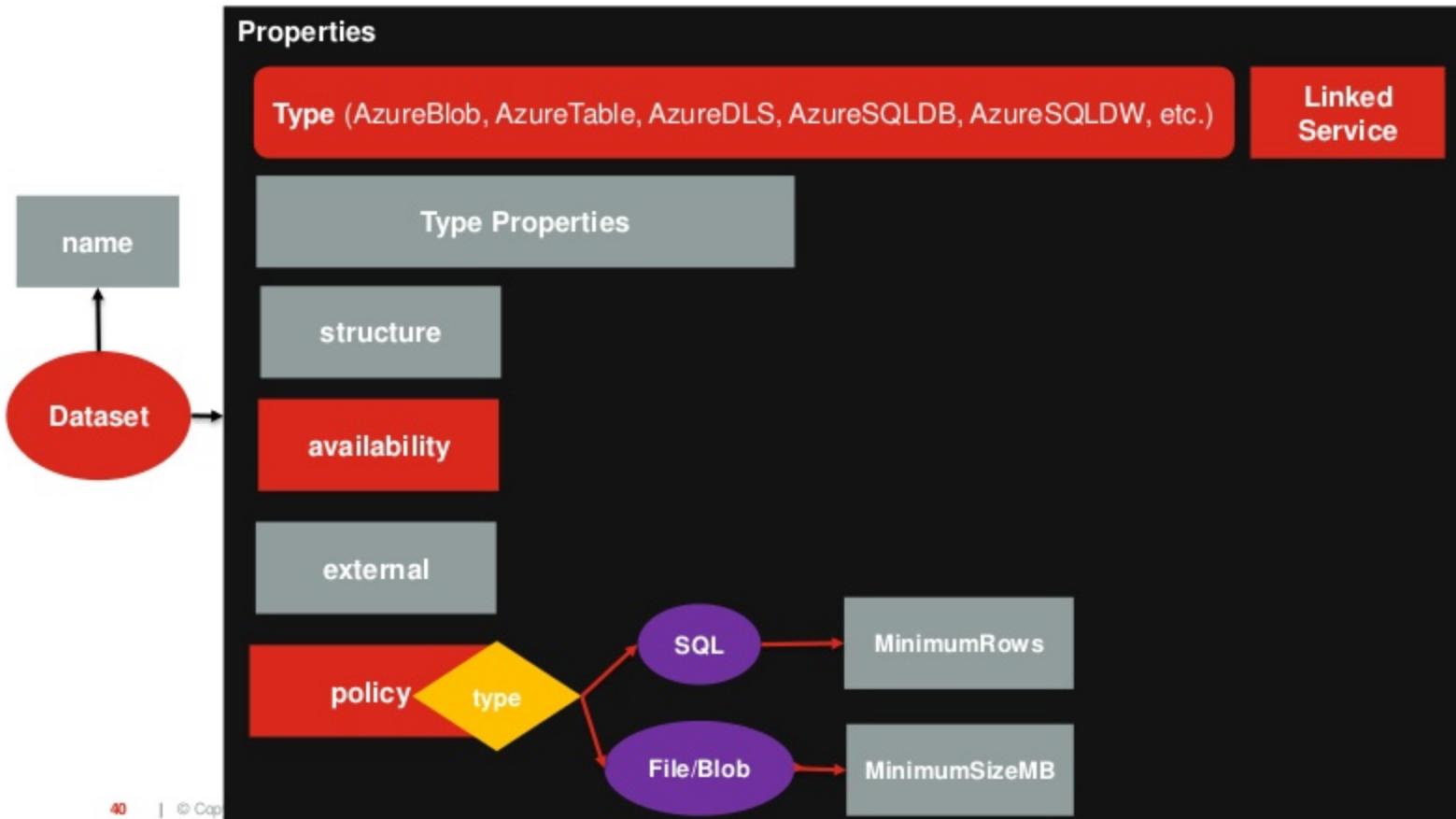
Azure Data Factory - Taxonomy

Dataset



Azure Data Factory - Taxonomy

Dataset



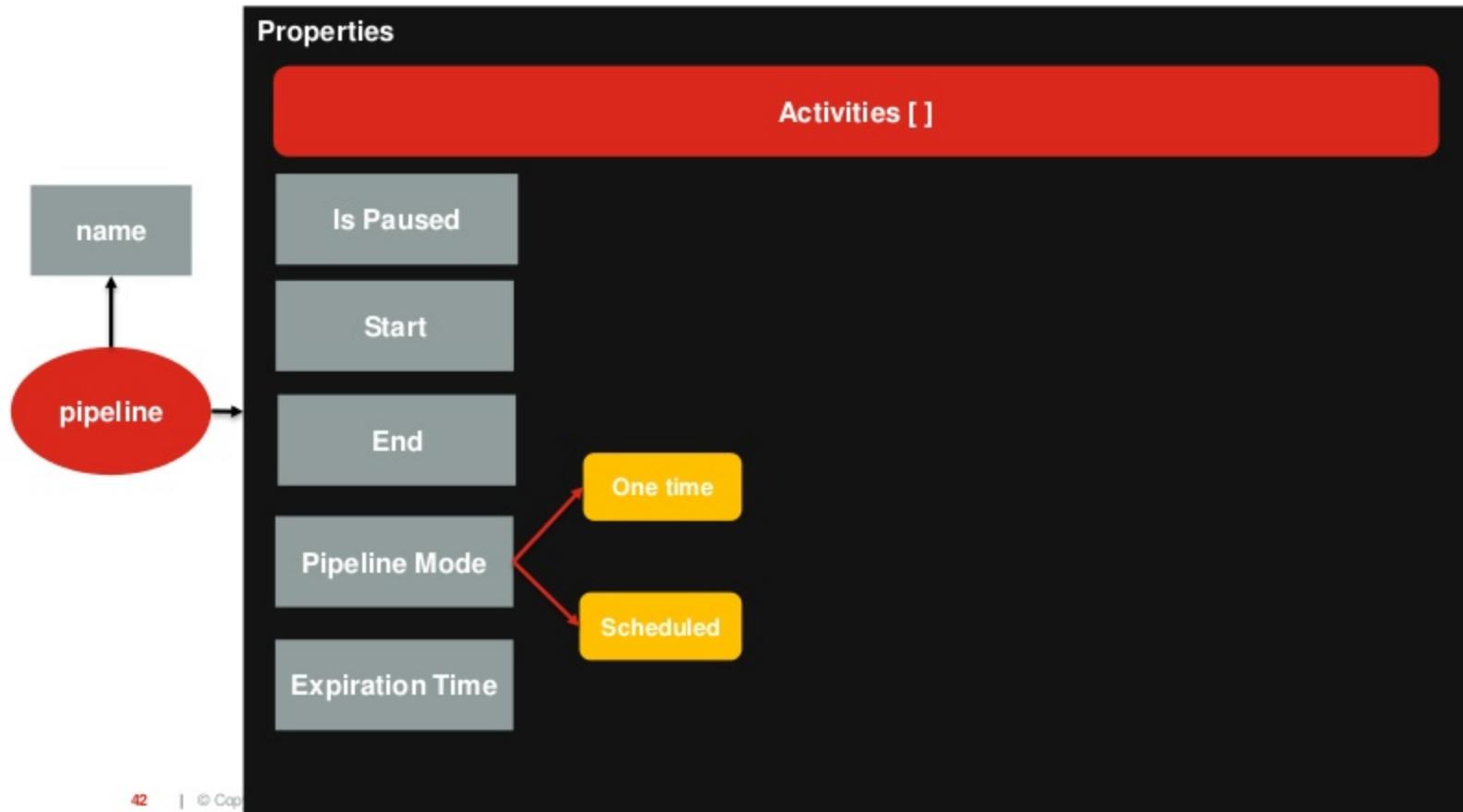
Azure Data Factory - Taxonomy

Pipeline

```
{  
    "name": "contoso_customer-pipeline",  
    "properties": {  
        "activities": [...],  
        "start": "2017-01-06T08:00:00.0106386Z",  
        "end": "2017-01-07T08:00:00.0106386Z",  
        "isPaused": false,  
        "pipelineMode": "Scheduled"  
    }  
}
```

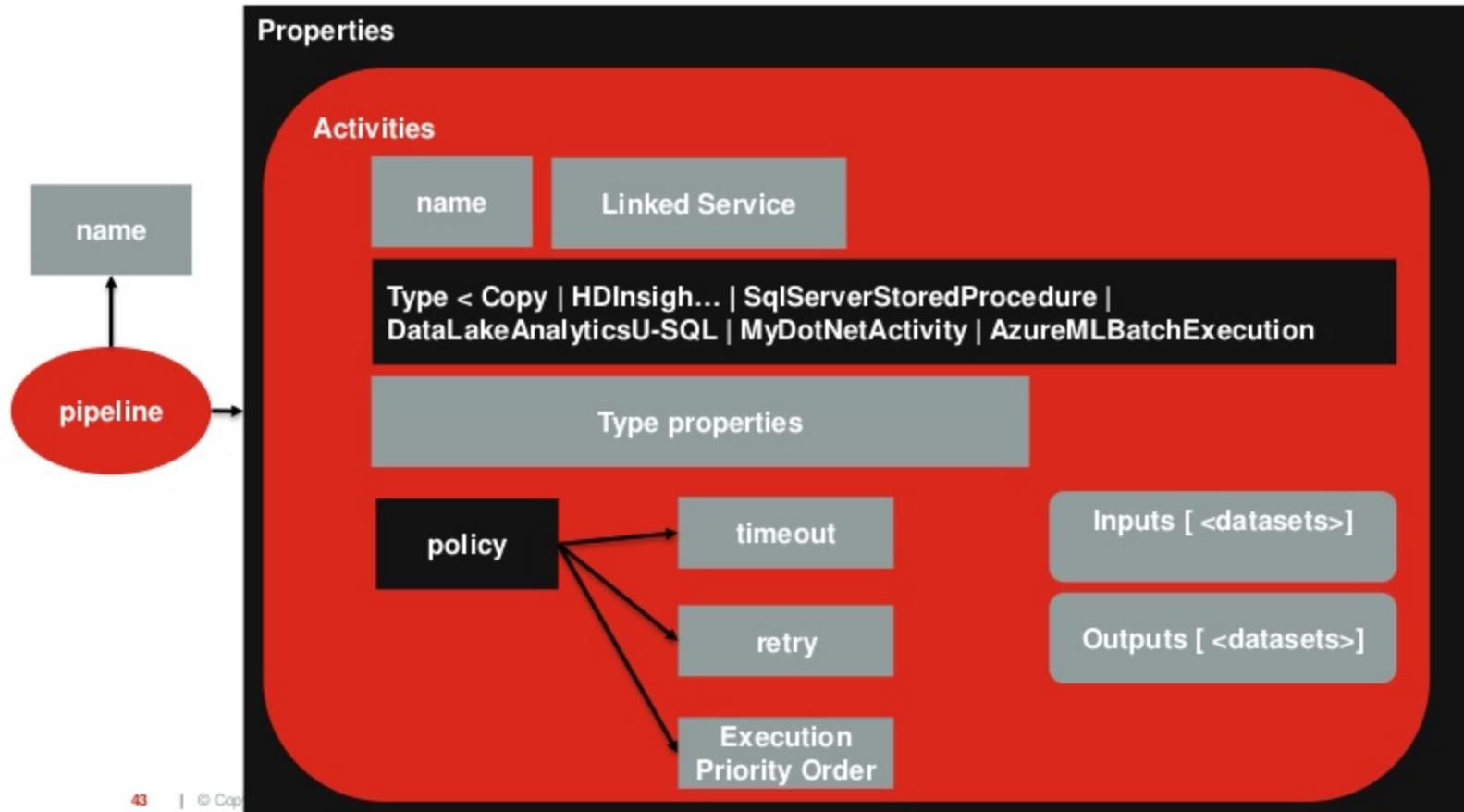
Azure Data Factory - Taxonomy

Pipeline



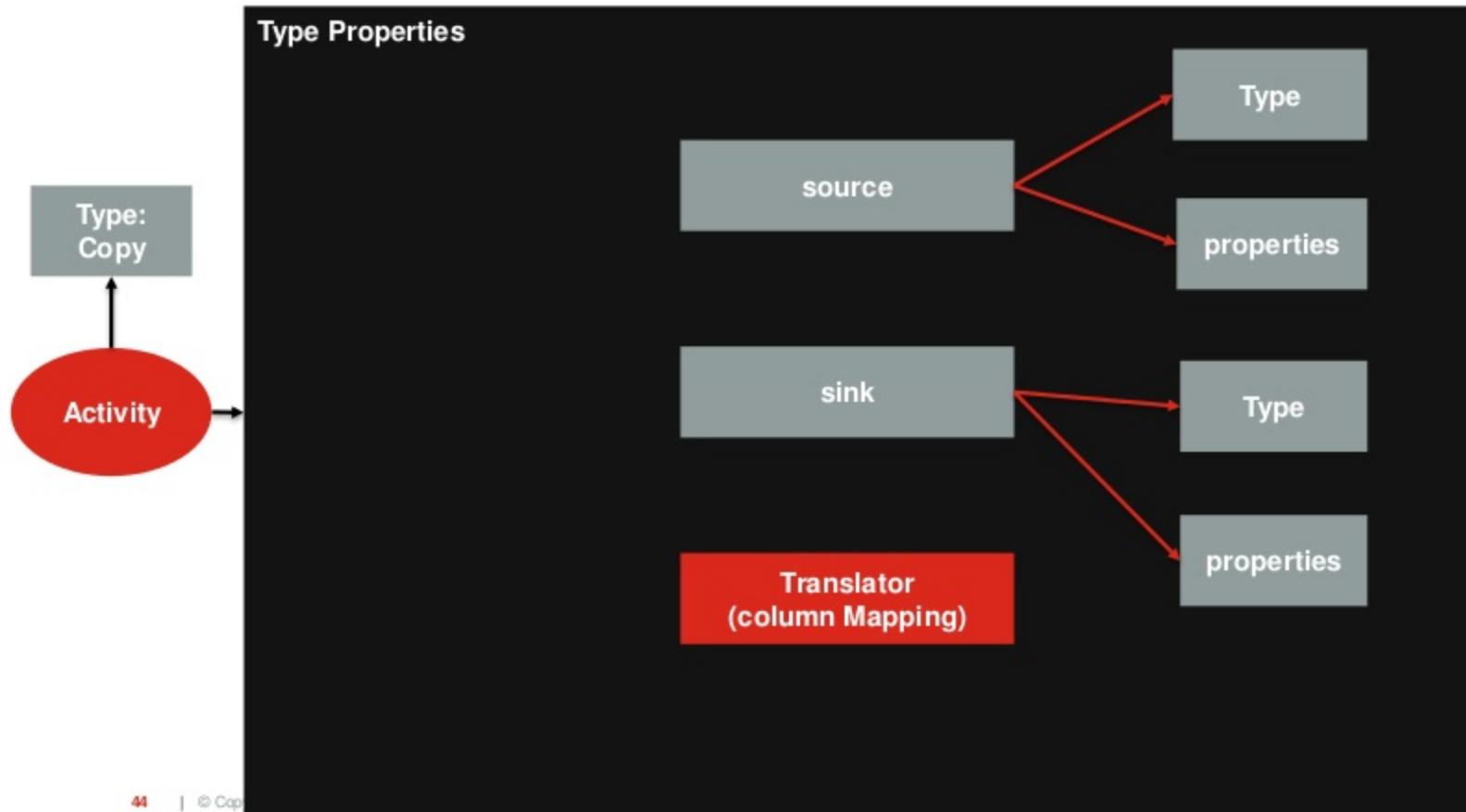
Azure Data Factory - Taxonomy

Pipeline - Activity



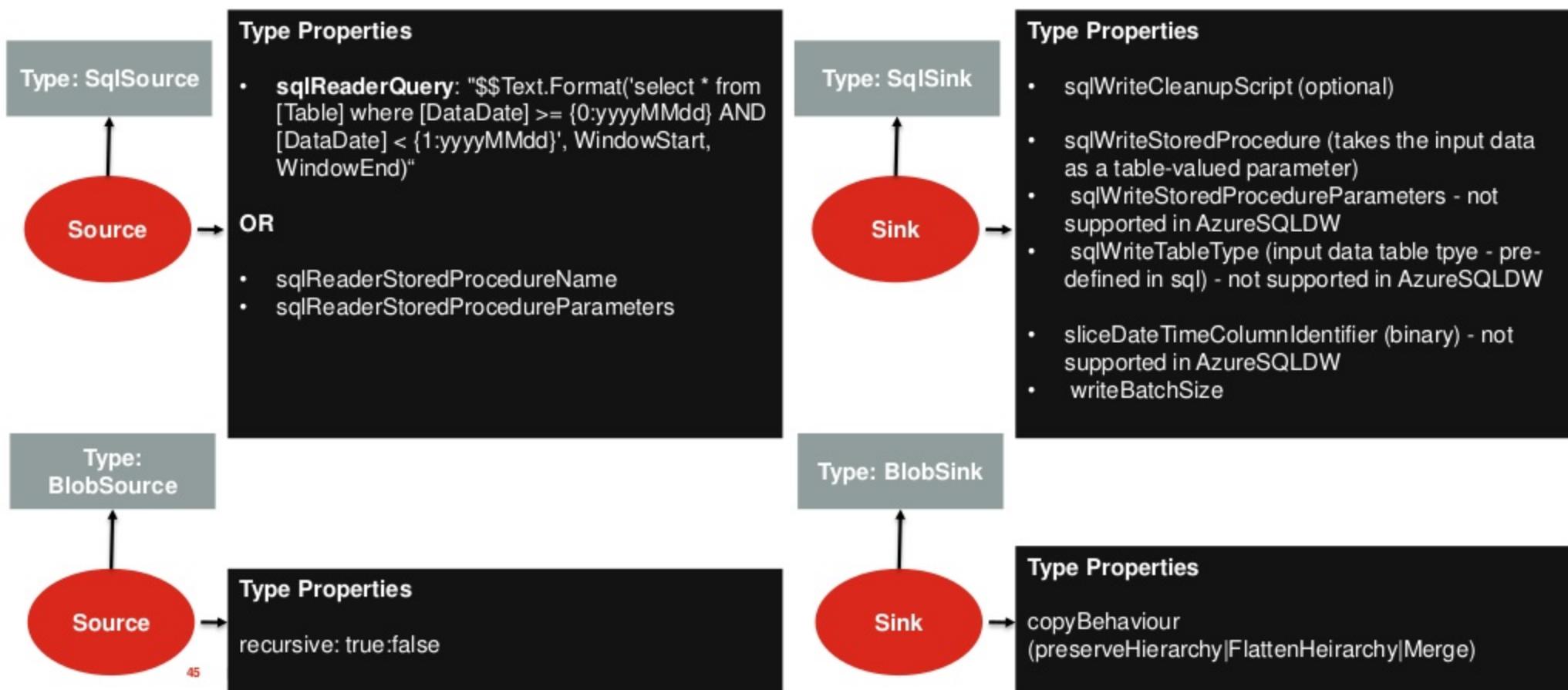
Azure Data Factory - Taxonomy

Data Movement Activity



Azure Data Factory - Taxonomy

Data Movement Activity – SQL & Blob



Azure Data Factory - Taxonomy

Copy Performance Tuning Elements

activity

concurrency

runs multiple data slices concurrently – data migration

parallelCopies

multiple threads, depending on source/sink

cloudMovementUnits

(1,2,4,8) - CPU/Memory allocation

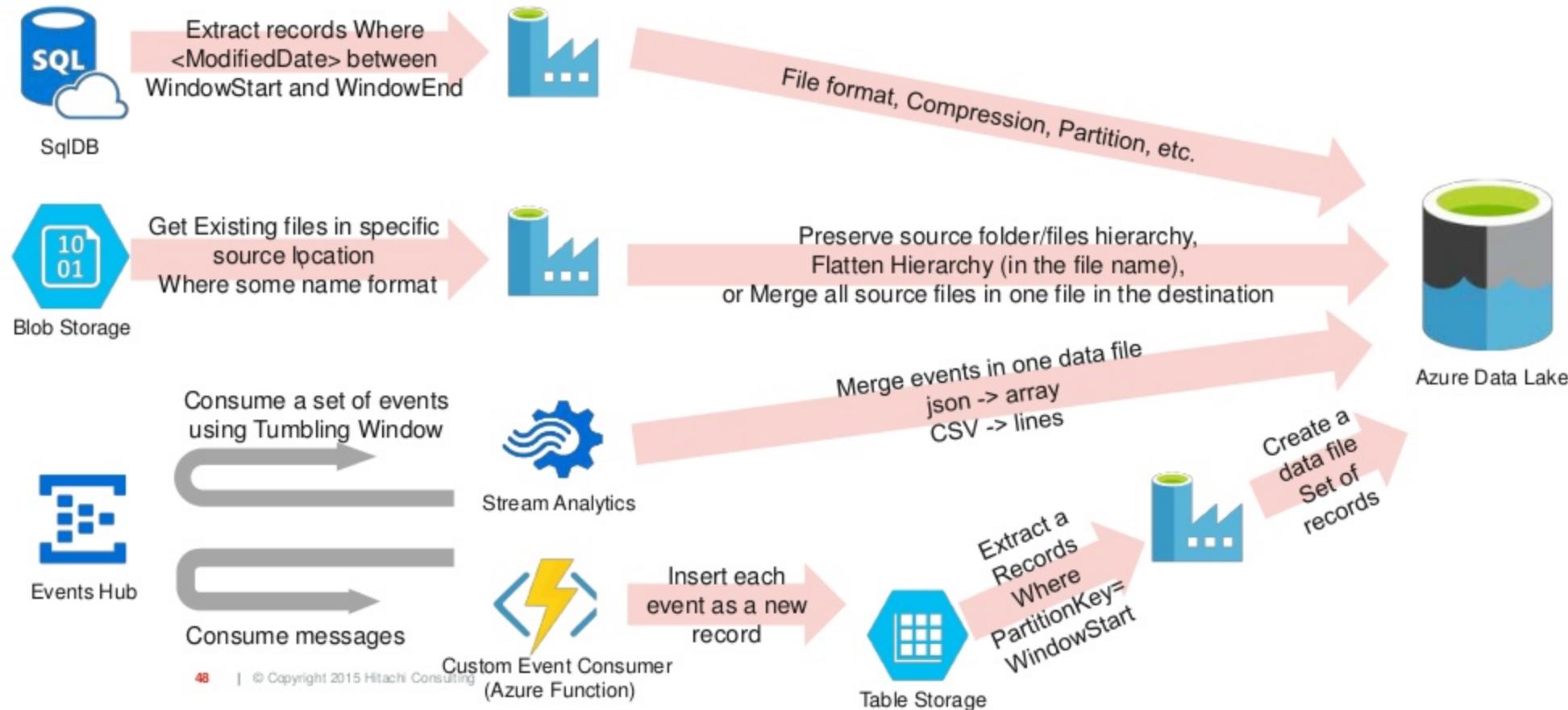
EnableStaging

Useful when moving data from on-prem to cloud

```
"enableStaging": true,  
"stagingSettings": {  
    "linkedServiceName": "MyStagingBlob",  
    "path": "stagingcontainer/path",  
    "enableCompression": true
```

Data Lake Ingestion Scenarios with Azure Data Factory

Azure Data Lake Ingestion Scenarios



Azure Data Lake Ingestion Scenarios

Data Factory - ADLS Dataset Definition

```
{  
    "name": "output-azdlsfolder-contoso_customer",  
    "properties": {  
        "published": false,  
        "type": "AzureDataLakeStore",  
        "linkedServiceName": "azdls_kspocs",  
        "typeProperties": {  
            "folderPath": "contoso/customers/{Year}/{Month}/{Day}",  
            "format": {  
                "type": "TextFormat", "rowDelimiter": "\n",  
                "columnDelimiter": "|", "nullValue": "NA",  
                "quoteChar": "\"", "firstRowAsHeader": true  
            },  
            "partitionedBy": [  
                {"name": "Year", "value": {"type": "DateTime", "date": "SliceStart", "format": "yyyy"}},  
                {"name": "Month", "value": {"type": "DateTime", "date": "SliceStart", "format": "MM"}},  
                {"name": "Day", "value": {"type": "DateTime", "date": "SliceStart", "format": "dd"}}  
            ],  
            "compression": {"type": "GZip", "level": "Optimal"}  
        },  
        "availability": {  
            "frequency": "Day",  
            "interval": 1  
        },  
        "external": false  
    }  
}
```



Define Destination File Format, Compression, and Partitioning - in the DL Store

Azure Data Lake Ingestion Scenarios

Data Factory – Copy Activity (SqlSource)

```

"activities": [
    {
        "type": "Copy",
        "typeProperties": {
            "source": {
                "type": "SqlSource",
                "sqlReaderQuery": "$$Text.Format('select * from cso.FactOnlineSales where CAST(CONVERT(VARCHAR(10), DateKey, 112)  
AS INT) >= {0:yyyyMMdd} AND CAST(CONVERT(VARCHAR(10), DateKey, 112) AS INT) < {1:yyyyMMdd}', WindowStart, WindowEnd)"
            },
            "sink": {
                "type": "AzureDataLakeStoreSink", "writeBatchSize": 0, "writeBatchTimeout": "00:00:00"
            }
        },
        "inputs": [{"name": "input-azsqltable-contoso_sales"}],
        "outputs": [{"name": "output-azdlsfolder-contoso_sales"}],
        "policy": {
            "timeout": "01:00:00",
            "concurrency": 1,
            "retry": 1
        },
        "scheduler": {
            "frequency": "Day",
            "interval": 1
        },
        "name": "act-copy-sales_SQLDB-DLS"
    }
]

```

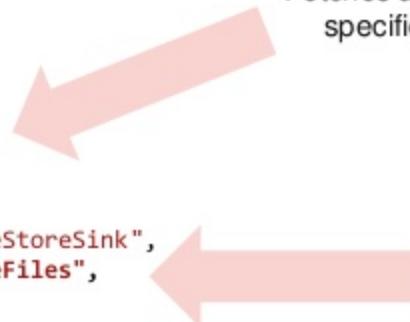


Parameterizing the fetch query to get only the current data slice (based on the WindowStart & WindowEnd)

Azure Data Lake Ingestion Scenarios

Data Factory – Copy Activity (BlobSource)

```
{  
  "name": "contoso_product-pipeline",  
  "properties": {  
    "activities": [  
      {  
        "type": "Copy",  
        "typeProperties": {  
          "source": {  
            "type": "BlobSource",  
            "recursive": true  
          },  
          "sink": {  
            "type": "AzureDataLakeStoreSink",  
            "copyBehavior": "MergeFiles",  
          }  
        },  
        "inputs": [{"name": "input-azstorage-contoso_product"}],  
        "outputs": [{"name": "output-azdlsfolder-contoso_product_v3"}],  
        "policy": {"timeout": "1.00:00:00", "concurrency": 1, "retry": 1},  
        "scheduler": {"frequency": "Day", "interval": 1},  
        "name": "act-copy-customer_AZS->DLS"  
      }  
    ],  
    "start": "2017-01-06T08:00:00.0106386Z",  
    "end": "2017-01-07T08:00:00.0106386Z",  
    "isPaused": false,  
    "pipelineMode": "Scheduled"  
  }  
}
```



Fetches all the files with in the input dataset specified path (folder) and sub folders

Merges all the fetched files data into one file to be stored in the destination – Other Options: PreserveHierarchy & Flatten

Azure Data Lake Ingestion Scenarios

Stream Analytics – From Events Hub

sa01-ingestdls-test

Query

Save Discard Test

Inputs (1)

- input-eventhub-test

Outputs (1)

- output-dls-streamdata

Need help with your query? Check out some of the most common Stream Analytics query patterns [here](#).

```
1 SELECT
2     Identifier,
3     ClickCount,
4     Basket,
5     COUNT(1) AS Dummy
6 INTO
7     [output-dls-streamdata]
8 FROM
9     [input-eventhub-test]
10 WHERE
11     Version = 'v2'
12 GROUP BY
13     Identifier,
14     ClickCount,
15     Basket,
16     TumblingWindow(Minute, 1)
17
```

Output details

output-dls-streamdata

Test Delete

* Account Name

* Path prefix pattern
Example: cluster1/logs/{date}/{time}

Date format

Time format

* Event serialization format

Encoding

Format

Azure Data Lake Analytics

Azure Data Lake Analytics

Big Data Processing Jobs

Big Data Processing Jobs as Service

Dynamic scaling

U-SQL: simple and familiar,
powerful, and extensible

Develop faster, debug,
and optimize smarter
using familiar tool

Deploy on Azure & schedule
using Data Factory

Integrates seamlessly with
your IT investments

Affordable and cost
effective: Pay as you use



Azure Data Lake Analytics

U-SQL

C# meets SQL
 (.NET data type, C# expressions + SQL set statements)

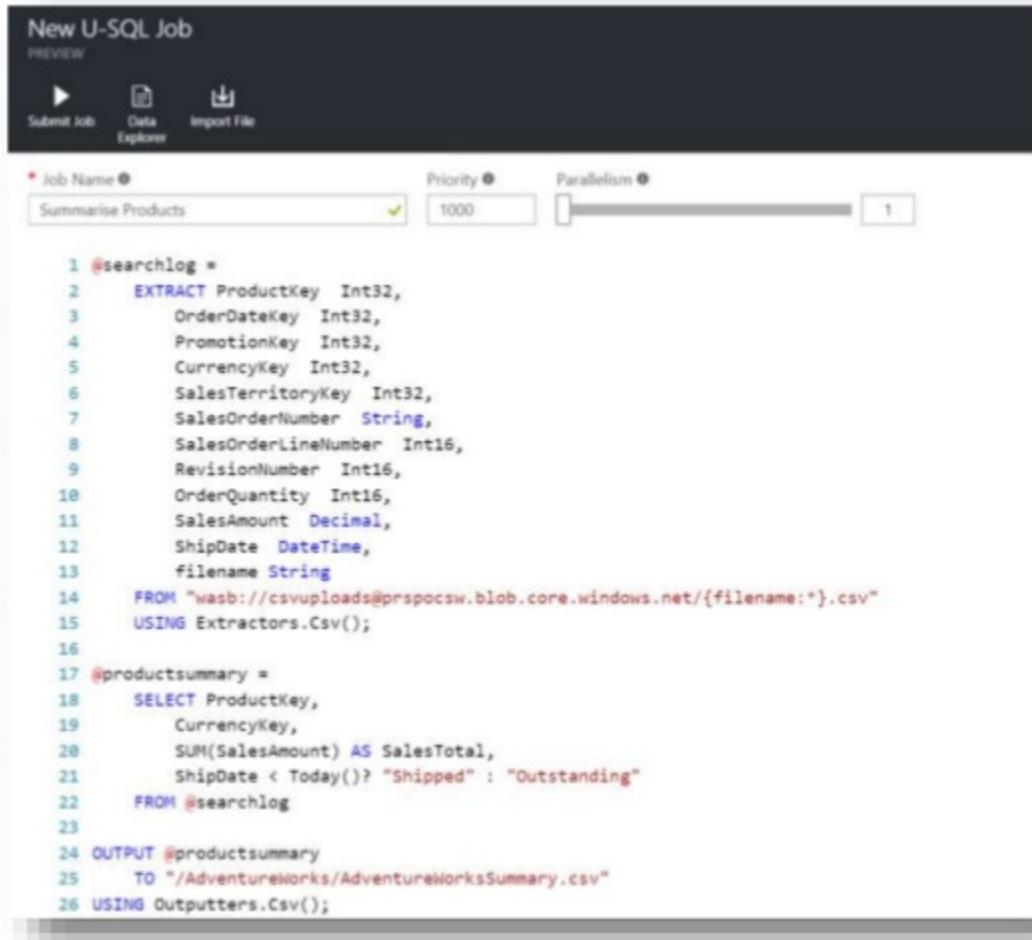
Works with structured and unstructured data

Set-based and procedural processing

Reuse of custom build .NET assemblies

Extensible (User-defined Functions, Operators and Aggregate Functions)

Natively Parallel



The screenshot shows the 'New U-SQL Job' interface. At the top, there are tabs for 'PREVIEW' (selected), 'Submit Job', 'Data Explorer', and 'Import File'. Below these are fields for 'Job Name' (set to 'Summarise Products'), 'Priority' (set to 1000), and 'Parallelism' (set to 1). The main area displays the U-SQL code:

```

1 @searchlog =
2     EXTRACT ProductKey Int32,
3         OrderDateKey Int32,
4             PromotionKey Int32,
5                 CurrencyKey Int32,
6                     SalesTerritoryKey Int32,
7                         SalesOrderNumber String,
8                             SalesOrderLineNumber Int16,
9                                 RevisionNumber Int16,
10                                OrderQuantity Int16,
11                                    SalesAmount Decimal,
12                                        ShipDate DateTime,
13                                            filename String
14 FROM "wasb://csvuploads@prspocsw.blob.core.windows.net/{filename:}.csv"
15 USING Extractors.Csv();
16
17 @productsummary =
18     SELECT ProductKey,
19         CurrencyKey,
20             SUM(SalesAmount) AS SalesTotal,
21                 ShipDate < Today()? "Shipped" : "Outstanding"
22             FROM @searchlog
23
24 OUTPUT @productsummary
25     TO "/AdventureWorks/AdventureWorksSummary.csv"
26 USING Outputters.Csv();

```

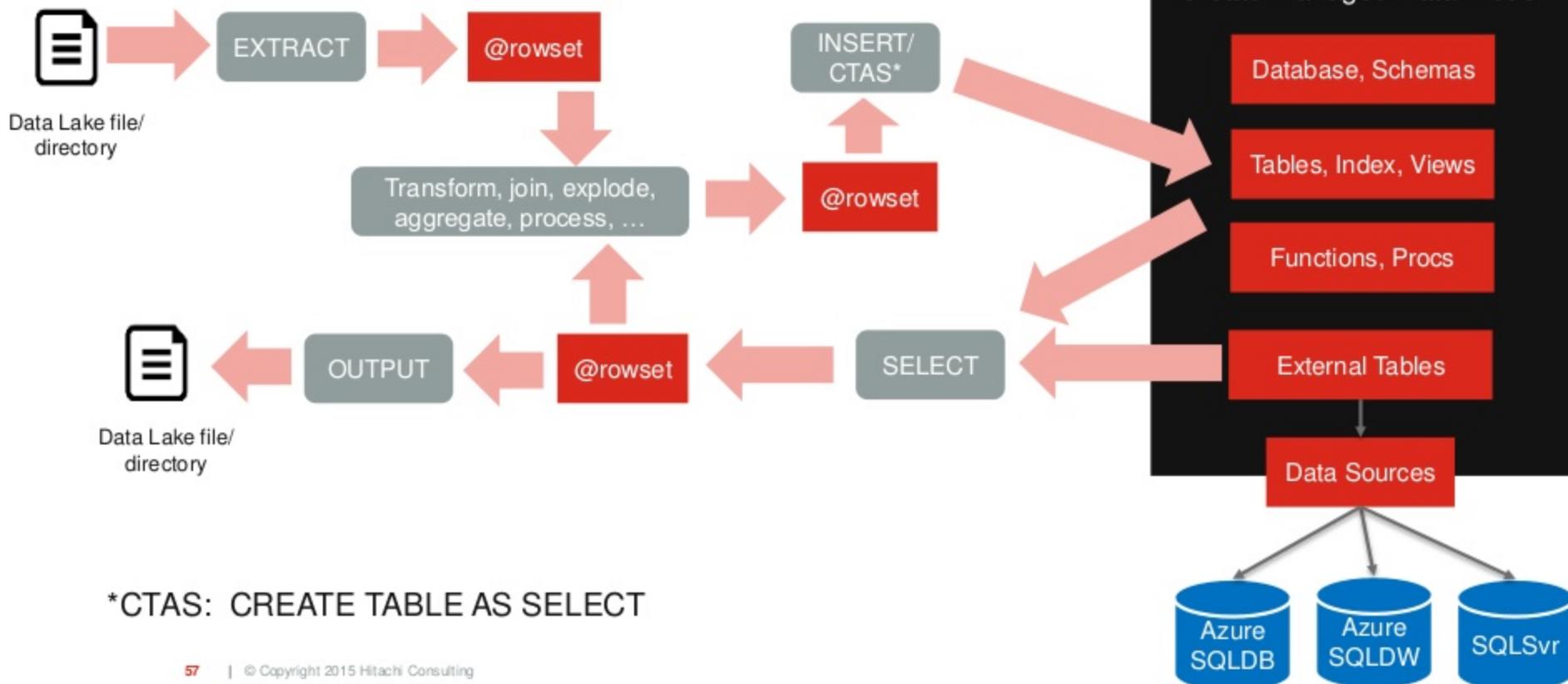
Azure Data Lake Analytics

U-SQL – Processing Model

- **Create a Managed Data Model** (database, schemas, tables, functions)
 - Only a schema on top of existing files
 - Allow querying data files using SQL on top with tables/views
 - Reuse of procs and functions
- **Retrieve data:**
 - From data lake files using **EXTRACT** – apply schema on read
 - From Managed table using **SELECT**
 - From external data sources (Azure SQL DB/DW, SQLServer)
- **Data is retrieved as @rowset**
- A rowset is processed (aggregated, joined with other rowset, etc.) using SQL and C# to produce another rowset
- A rowset is stored:
 - As data lake file using OUTPUT
 - In managed table using **INSERT** (apply schema on write) or **CREATE TABLE AS SELECT**

Azure Data Lake Analytics

U-SQL – Processing Model



*CTAS: CREATE TABLE AS SELECT

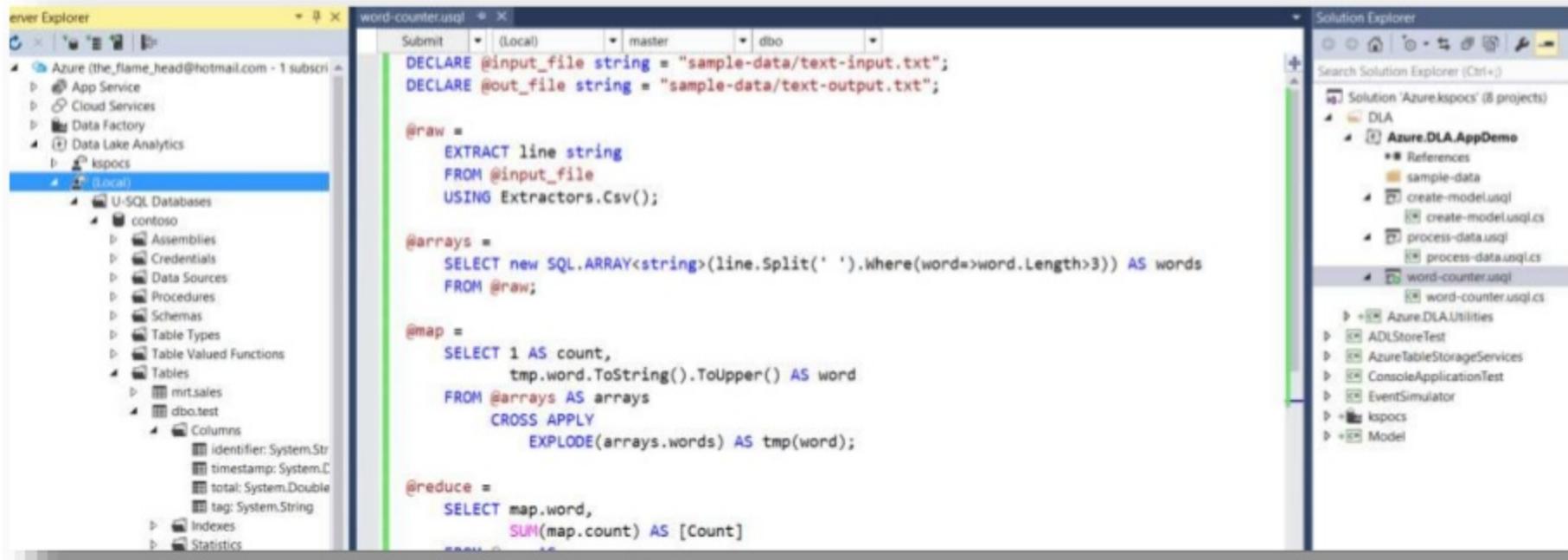
Azure Data Lake Analytics

U-SQL – Script Structure

- Use statement: **USE Database**
- Declare Variables: **DECLARE @input string = <File Location>**
- Register Assembly: **CREATE ASSEMBLY <myDotNetLib> FROM "<dll location>"**
- Reference Assembly: **REFERENCE ASSEMBLY <myDotNetLib>**
- DLL Statement: **CREATE TABLE | VIEW |...**
→ Create Table: **Index, Partition, Distribution**
- Query statement: **@rowset = SELECT ... FROM <@set>|<table>**
- Extract statement: **@rowset = EXTRACT <columns> FROM <file location> USING <Extractor>**
- DML Statement: **INSERT INTO <Table> SELECT ... FROM @rowset**
- CTAS: **CREATE TABLE <Table> AS SELECT ... FROM @rowset**
- Output Statement: **OUTPUT @rowset TO <file location> USING <Outputer>**

Azure Data Lake Analytics

Dev and Test U-SQL locally (using Visual Studio Data Lake Analytics Tools)



The screenshot shows the Visual Studio interface for developing U-SQL scripts. The left pane is the Server Explorer, displaying Azure resources like App Service, Cloud Services, Data Factory, and Data Lake Analytics. The center pane is the U-SQL Editor with the script content, and the right pane is the Solution Explorer showing the project structure.

```

DECLARE @input_file string = "sample-data/text-input.txt";
DECLARE @out_file string = "sample-data/text-output.txt";

@raw =
    EXTRACT line string
    FROM @input_file
    USING Extractors.Csv();

@arrays =
    SELECT new SQL.ARRAY<string>(line.Split(' ')).Where(word=>word.Length>3) AS words
    FROM @raw;

@map =
    SELECT 1 AS count,
        tmp.word.ToString().ToUpper() AS word
    FROM @arrays AS arrays
    CROSS APPLY
        EXPLODE(arrays.words) AS tmp(word);

@reduce =
    SELECT map.word,
        SUM(map.count) AS [Count]
    FROM @map
    GROUP BY map.word;
  
```

Solution Explorer content:

- Solution 'Azure.kspocs' (8 projects)
 - DLA
 - Azure.DLA.AppDemo
 - References
 - sample-data
 - create-model.usql
 - process-data.usql
 - process-data.usql.cs
 - word-counter.usql
 - word-counter.usql.cs
 - ADLStoreTest
 - AzureTableStorageServices
 - ConsoleApplicationTest
 - EventSimulator
 - kspocs
 - Model

Azure Data Lake Analytics

U-SQL Example – data aggregation

Create Managed Model

```
USE DATABASE master;

CREATE DATABASE IF NOT EXISTS contoso;

USE DATABASE contoso;

CREATE SCHEMA IF NOT EXISTS mrt;

DROP TABLE IF EXISTS mrt.sales;

CREATE TABLE mrt.sales
(
    Year int,
    Month int,
    Country string,
    ProductSubCategory string,
    TotalSales double,

    INDEX index_sales CLUSTERED(ProductSubCategory ASC)
    PARTITIONED BY (Year,Month)
    DISTRIBUTED BY HASH (Country)
);
```

Proccess Data

```
DECLARE @datafolder string = "sample-data/data-files/*.txt";

@data =
    EXTRACT
        identifier string,
        value      double
    FROM @datafolder
    USING Extractors.Csv();

@result =
    SELECT
        identifier,
        DateTime.Now AS timestamp,
        SUM(value) AS total
    FROM @data
    GROUP BY
        identifier,
        DateTime.Now;

USE DATABASE contoso;

DROP TABLE IF EXISTS dbo.test;

CREATE TABLE dbo.test
(
    INDEX [idx_test] CLUSTERED (identifier ASC)
    PARTITIONED BY HASH (identifier)
)
AS
SELECT
    *
FROM @result;
```

Azure Data Lake Analytics

U-SQL Example – data aggregation

Screencast showing the Azure Data Lake Analytics U-SQL interface.

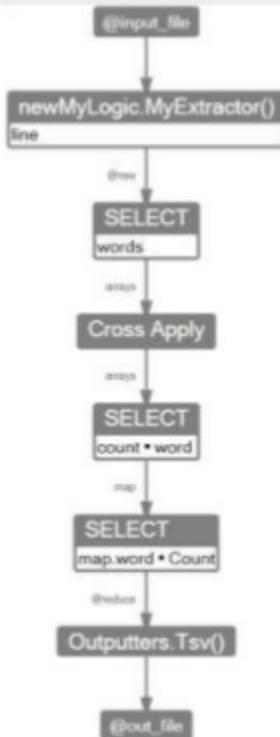
The interface includes:

- Explorer:** Shows the Azure subscription, App Service, Cloud Services, Data Factory, and Data Lake Analytics environment. The U-SQL Databases node is expanded, showing contoso, master, and Assemblies.
- Table Preview:** For the test table, showing the schema (identifier, timestamp, total, tag) and data (4 rows).
- Compile View:** C:\...\process-data.usql
- process-data.usql:** The U-SQL script for data processing.
- MyExtractor.cs:** A C# class for data extraction.
- Table Preview:** Shows the schema and data for the test table.
- Partition Table:** Shows the partitioned structure.
- Partition Chart:** Shows the distribution of data across partitions.
- Row Filter:** Row Filter: column1 == "value"; Column Filter: column1, column2, column3
- Select Row Count:** 100
- Filter:**
- Note:** Your query could be very slow if there is no CLUSTER or SORT key in your filter. Please refer to the schema for key information.
- Total 4 rows in test:**
- Encoding:** Unicode (UTF-8)
- Include column header:** [Copy to Clipboard](#)

identifier	timestamp	total	tag
1 c	2017-02-16T20:27	1245	
2 f	2017-02-16T20:27	1245	
3 b	2017-02-16T20:27	1245	
4 a	2017-02-16T20:27	690	

Azure Data Lake Analytics

U-SQL – Example Word Count



```

DECLARE @input_file string = "sample-data/text-input.txt";
DECLARE @out_file string = "sample-data/text-output.txt";

@raw =
    EXTRACT line string
    FROM @input_file
    USING Extractors.Csv();

@arrays =
    SELECT new SQL.ARRAY<string>(line.Split(' ')).Where(word=>word.Length>1)) AS
words
    FROM @raw;

@map =
    SELECT 1 AS count,
           tmp.word.ToString().ToUpper() AS word
    FROM @arrays AS arrays
    CROSS APPLY
        EXPLODE(arrays.words) AS tmp(word);

@reduce =
    SELECT map.word,
           SUM(map.count) AS [Count]
    FROM @map AS map
    GROUP BY map.word;

OUTPUT @reduce
TO @out_file
ORDER BY [Count] DESC
USING Outputters.Tsv();
  
```

U-SQL .net Extensibility

Azure Data Lake Analytics

U-SQL – .net extensibility

- **Code-behind** (script.usql.cs) – Write Static Methods and use in the u-sql script
- **Assemblies**
 - Create dlls with processing logic
 - Upload dlls to dls
 - Register assemblies in a data lake managed database
 - Reference assembly in u-sql script and use code

Implement custom:

- Extractors
- Outputters
- Aggregators
- Appliers
- Combiners
- Processor

Azure Data Lake Analytics

U-SQL – .net extensibility | terms frequency example

Managed Model

```

USE DATABASE master;

CREATE DATABASE IF NOT EXISTS demo;

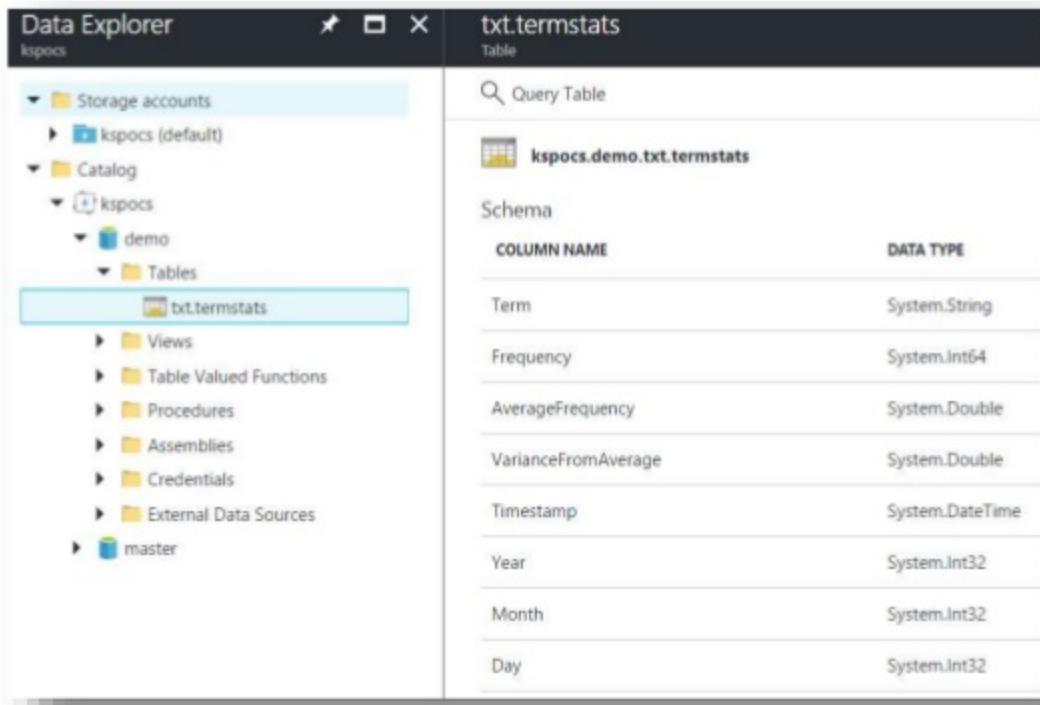
USE DATABASE demo;

CREATE SCHEMA IF NOT EXISTS txt;

DROP TABLE IF EXISTS txt.termstats;

CREATE TABLE txt.termstats
(
    Term string,
    Frequency long?,
    AverageFrequency double,
    VarianceFromAverage double,
    Timestamp DateTime,
    Year int,
    Month int,
    Day int,

    INDEX index_termstats CLUSTERED(Term ASC)
    PARTITIONED BY (Year,Month,Day)
    DISTRIBUTED BY HASH (Term)
);
    
```



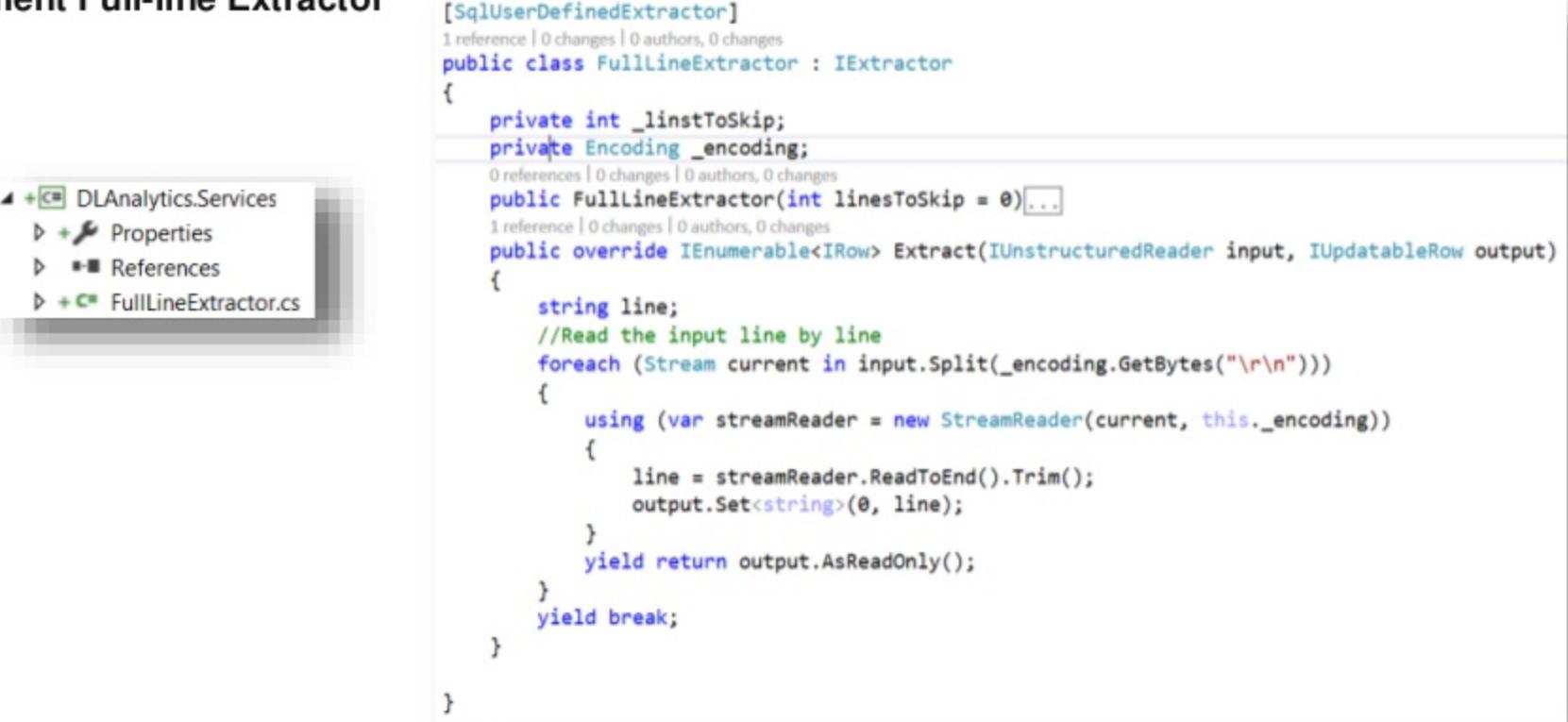
The screenshot shows the Azure Data Explorer interface with two panes. The left pane, titled 'Data Explorer' and 'kspocs', displays a tree view of database objects. It shows 'Storage accounts', 'kspocs (default)', 'Catalog', 'kspocs', 'demo', and 'Tables'. Under 'Tables', 'txt.termstats' is selected and highlighted with a green border. The right pane, titled 'txt.termstats' and 'Table', shows the schema of the 'txt.termstats' table. It includes a search bar, a 'Query Table' button, and a table with columns: COLUMN NAME and DATA TYPE. The schema is as follows:

COLUMN NAME	DATA TYPE
Term	System.String
Frequency	System.Int64
AverageFrequency	System.Double
VarianceFromAverage	System.Double
Timestamp	System.DateTime
Year	System.Int32
Month	System.Int32
Day	System.Int32

Azure Data Lake Analytics

U-SQL – .net extensibility / terms frequency example

Implement Full-line Extractor



The screenshot shows a Visual Studio code editor with the 'FullLineExtractor.cs' file open. The file is part of a project named 'DLAnalytics.Services'. The code implements an 'IExtractor' interface, specifically a 'FullLineExtractor'. It reads input line by line and sets each line as a single column in the output row. The code uses a StreamReader to read each line and trim it before setting it to the output row.

```
[SqlUserDefinedExtractor]
1 reference | 0 changes | 0 authors, 0 changes
public class FullLineExtractor : IExtractor
{
    private int _linesToSkip;
    private Encoding _encoding;
    public FullLineExtractor(int linesToSkip = 0)...
    public override IEnumerable<IRow> Extract(IUnstructuredReader input, IUpdatableRow output)
    {
        string line;
        //Read the input line by line
        foreach (Stream current in input.Split(_encoding.GetBytes("\r\n")))
        {
            using (var streamReader = new StreamReader(current, this._encoding))
            {
                line = streamReader.ReadToEnd().Trim();
                output.Set<string>(0, line);
            }
            yield return output.AsReadOnly();
        }
        yield break;
    }
}
```

Azure Data Lake Analytics

U-SQL – .net extensibility | terms frequency example

Registering Assembly

register-assemblies.usql

```
DECLARE @ASSEMBLY_PATH string = "resources/demo-libraries/";
DECLARE @DLAServices string = @ASSEMBLY_PATH+ "DLAnalytics.Services.dll";

CREATE DATABASE IF NOT EXISTS demo;

USE DATABASE demo;

DROP ASSEMBLY IF EXISTS DLAServices;
CREATE ASSEMBLY DLAServices
FROM @DLAServices;
```

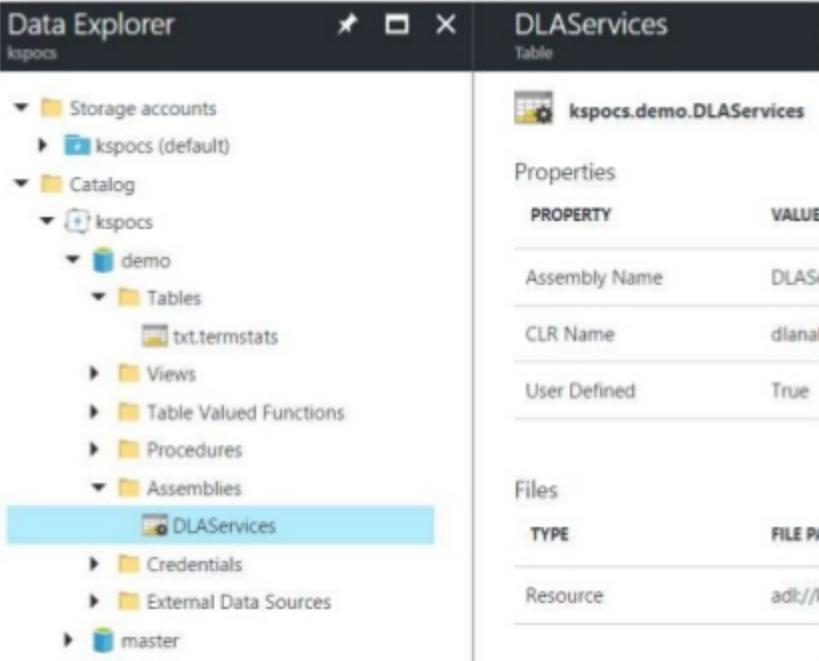
Reference, and use the Assembly

process-termstats.usql

```
USE DATABASE demo;

REFERENCE ASSEMBLY DLAServices;

USING MyServices = DLAnalytics.Services;
```



The screenshot shows the Data Explorer and DLAServices Table in the Azure Data Studio interface.

Data Explorer:

- Storage accounts
- lspocs (default)
- Catalog
- lspocs
 - demo
 - Tables
 - txt.termstats
 - Views
 - Table Valued Functions
 - Procedures
 - Assemblies
 - DLAServices
 - Credentials
 - External Data Sources
 - master

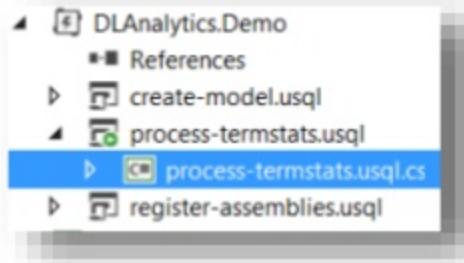
DLAServices Table:

PROPERTY	VALUE
Assembly Name	DLAS
CLR Name	dlanal
User Defined	True
Files	
TYPE	FILE P
Resource	adl://i

Azure Data Lake Analytics

U-SQL – .net extensibility | terms frequency example

Code-behind



```
namespace DLAnalytics.Demo
{
    0 references | 0 changes | 0 authors, 0 changes
    public static class Tokenizer
    {
        private static List<string> StopWords = new List<string>(
            "a has be here an and the this on with that to other others th
            1 reference | 0 changes | 0 authors, 0 changes
            public static string RemoveStopWords(string text)...
            0 references | 0 changes | 0 authors, 0 changes
            public static IEnumerable<string> Tokenize(int n, string text)
            {
                text = RemoveStopWords(text);

                var nGrams = new StringBuilder();
                var queue = new Queue<int>();
                int wordCount = 0;
                int lastWordLength = 0;

                if (!string.IsNullOrEmpty(text) && char.IsLetterOrDigit(text[0])
                {
                    nGrams.Append(text[0]);
                    lastWordLength++;
                }

                for (int charIndex = 1; charIndex < text.Length-1; charIndex++)
                {
                    char before = text[charIndex - 1];
                    char after = text[charIndex + 1];
```

Azure Data Lake Analytics

U-SQL – .net extensibility | term frequency example

process-termstats.usql

```

DECLARE @input_file string = "demo-data/doc1.txt";

USE DATABASE demo;

REFERENCE ASSEMBLY DLAzure;
USING MyServices = DLAzure.Services;

@raw =
    EXTRACT line string
    FROM @input_file
    USING new MyServices.FullLineExtractor();

@arrays =
    SELECT new
SQL.ARRAY<string>(DLAnalytics.Demo.Tokenizer.Tokenize(2,line))
AS terms FROM @raw;

@map =
    SELECT 1 AS frequency, tmp.term.ToUpper() AS term
    FROM @arrays AS arrays
    CROSS APPLY
        EXPLODE(arrays.terms) AS tmp(term);

@reduce =
    SELECT map.term,
        SUM(map.frequency) AS frequency
    FROM @map AS map
    GROUP BY map.term
    HAVING SUM(map.frequency) >= 3;

```

```

@aggregates =
    SELECT Convert.ToDouble(AVG(frequency)) AS averageFrequency
    FROM @reduce;

@output =
    SELECT term,
        frequency,
        averageFrequency,
        Convert.ToDouble(frequency - averageFrequency) AS varianceFromAverage,
        DateTime.Now AS timestamp
    FROM @reduce CROSS JOIN @aggregates;

DECLARE @Year int = DateTime.Now.Year;
DECLARE @Month int = DateTime.Now.Month;
DECLARE @Day int = DateTime.Now.Day;

ALTER TABLE txt.termstats
ADD IF NOT EXISTS PARTITION(@Year, @Month, @Day);

INSERT INTO txt.termstats
(Term, Frequency, AverageFrequency, VarianceFromAverage, Timestamp )
PARTITION
(
    @Year,
    @Month,
    @Day
)
SELECT *
FROM @output;

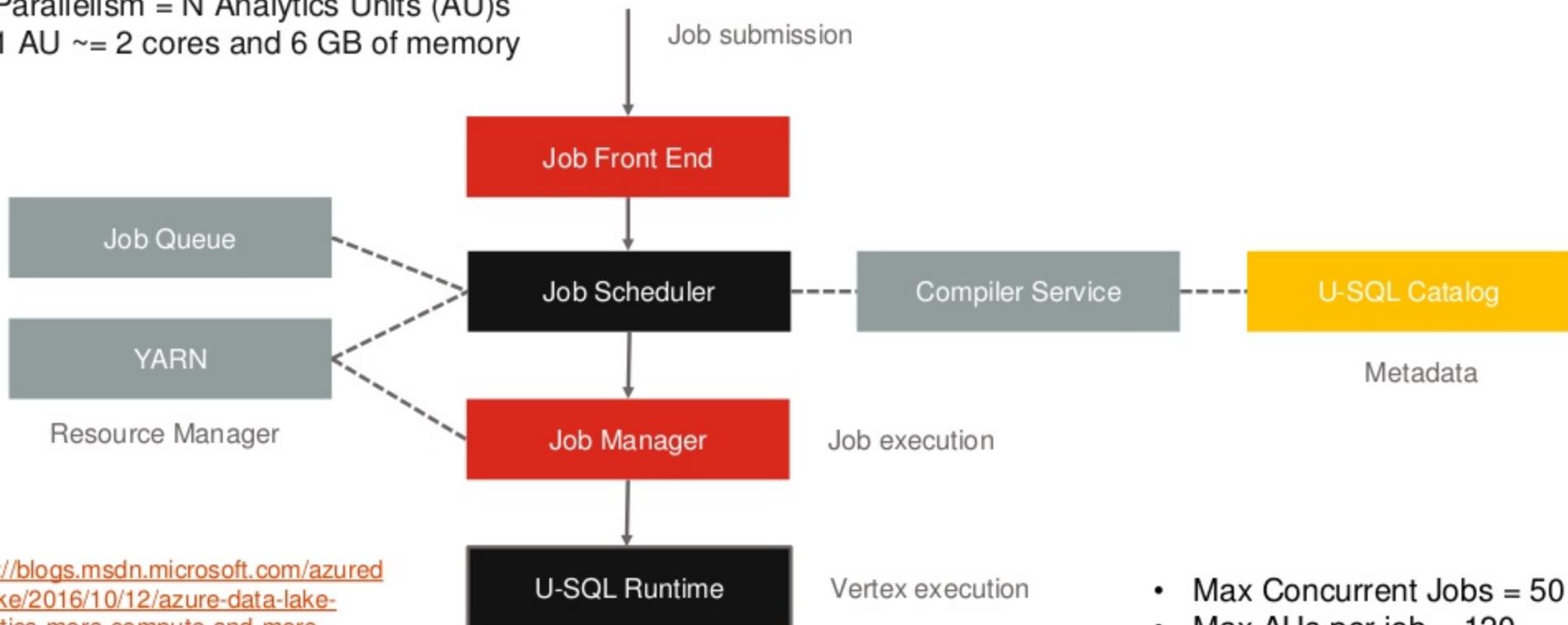
```

Running U-SQL Jobs on Azure

Azure U-SQL Jobs

Simplified Workflow

- Parallelism = N Analytics Units (AU)s
- 1 AU ≈ 2 cores and 6 GB of memory

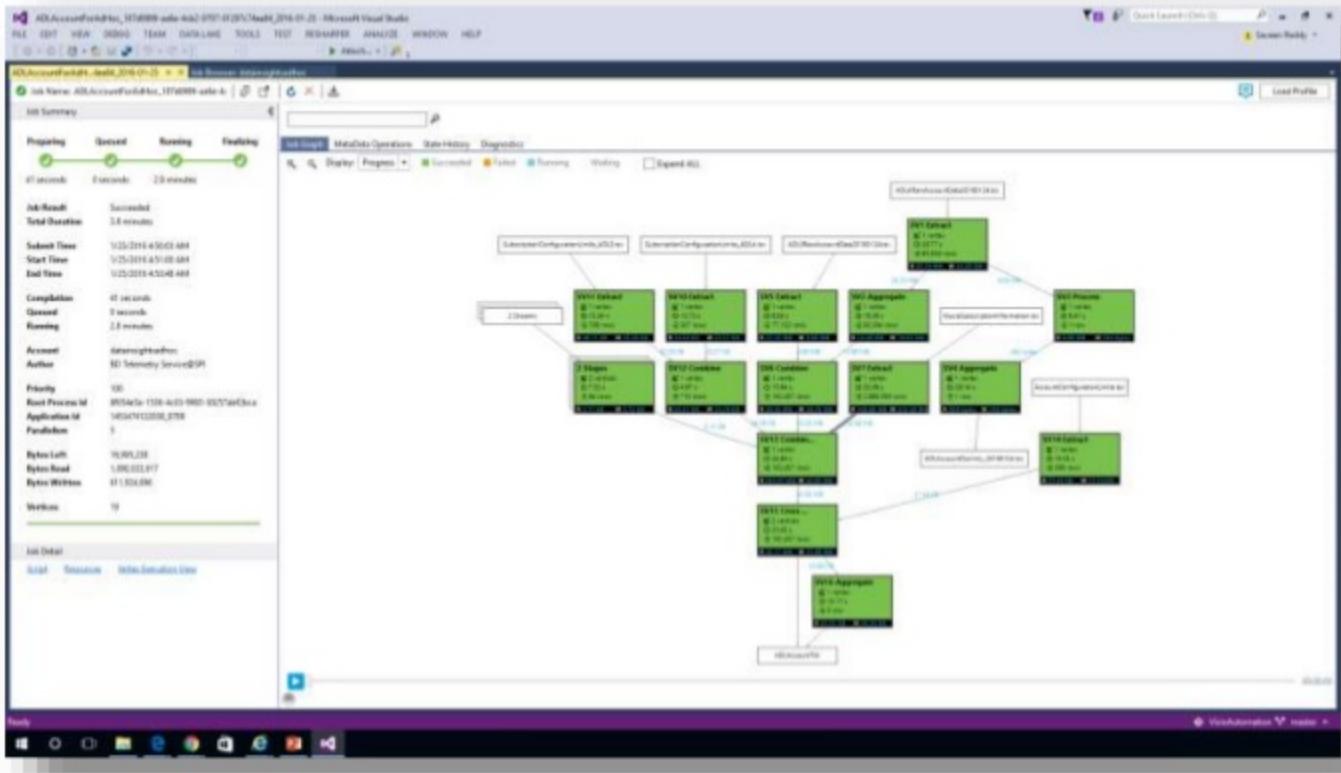


<https://blogs.msdn.microsoft.com/azuredatalake/2016/10/12/azure-data-lake-analytics-more-compute-and-more-control>

- Max Concurrent Jobs = 50
- Max AUs per job = 120
- Max Queue Length = 200

Azure U-SQL Jobs

Monitoring Jobs running on Azure



Job Summary

- Preparing: 37 seconds
- Queued: 0 seconds
- Running: 9.9 minutes
- Finalizing: 0 seconds

Job Result: Succeeded
Total Duration: 10.8 minutes

Submit Time: 1/24/2016 10:14:52 AM
Start Time: 1/24/2016 10:15:48 AM
End Time: 1/24/2016 10:25:42 AM

Compilation: 37 seconds
Queued: 0 seconds
Running: 9.9 minutes

Account: datainsightsadhoc
Author: BD Telemetry Service@SPI

Priority: 100
Root Process ID: 9854fe3e-1108-4c03-9801-927fae8ca
Application ID: 14524713308.2798
Parallelism: 5

Bytes Left: 76,893,238
Bytes Read: 1,096,023,917
Bytes Written: 81,705,086

Workers: 19

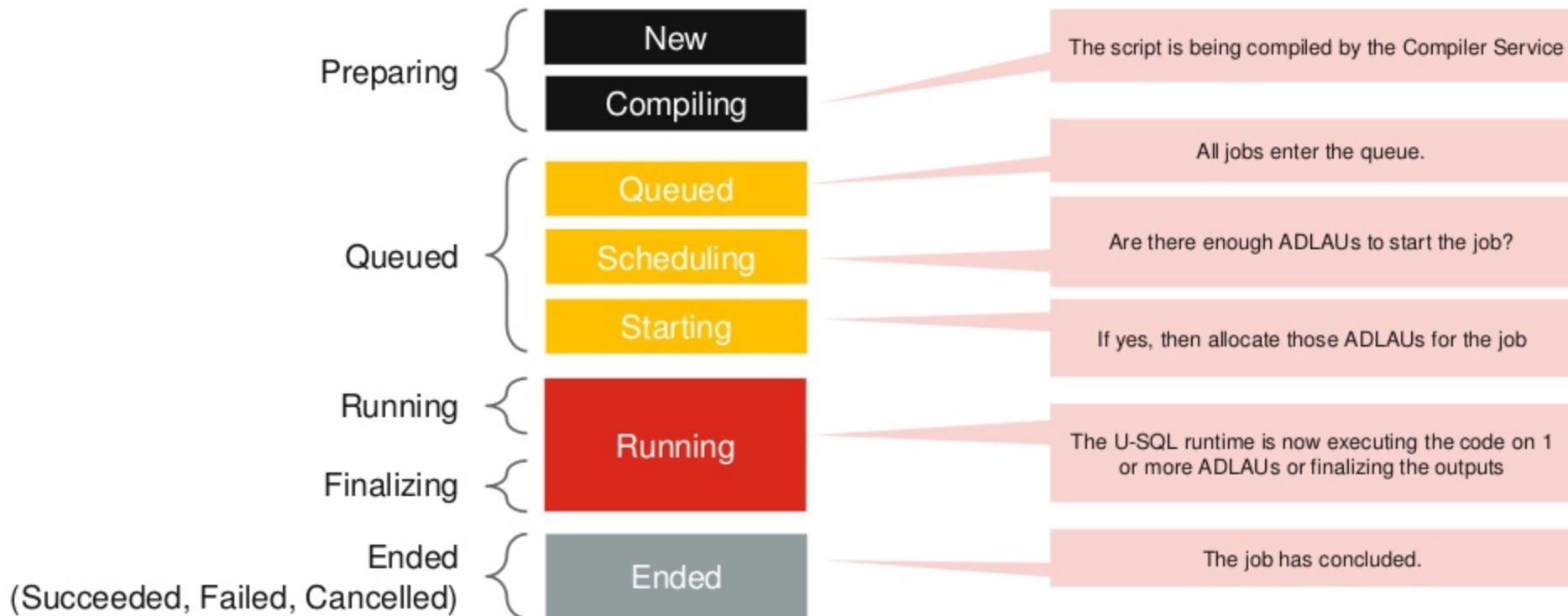
Job Detail

Job Summary

Preparing	Queued	Running	Finalizing
✓	✓	✓	✓
37 seconds	0 seconds	9.9 minutes	0 seconds
Job Result			Succeeded
Total Duration			10.8 minutes
Submit Time			1/24/2016 10:14:52 AM
Start Time			1/24/2016 10:15:48 AM
End Time			1/24/2016 10:25:42 AM
Compilation			37 seconds
Queued			0 seconds
Running			9.9 minutes
Account			datainsightsadhoc
Author			BD Telemetry Service@SPI

Azure U-SQL Jobs

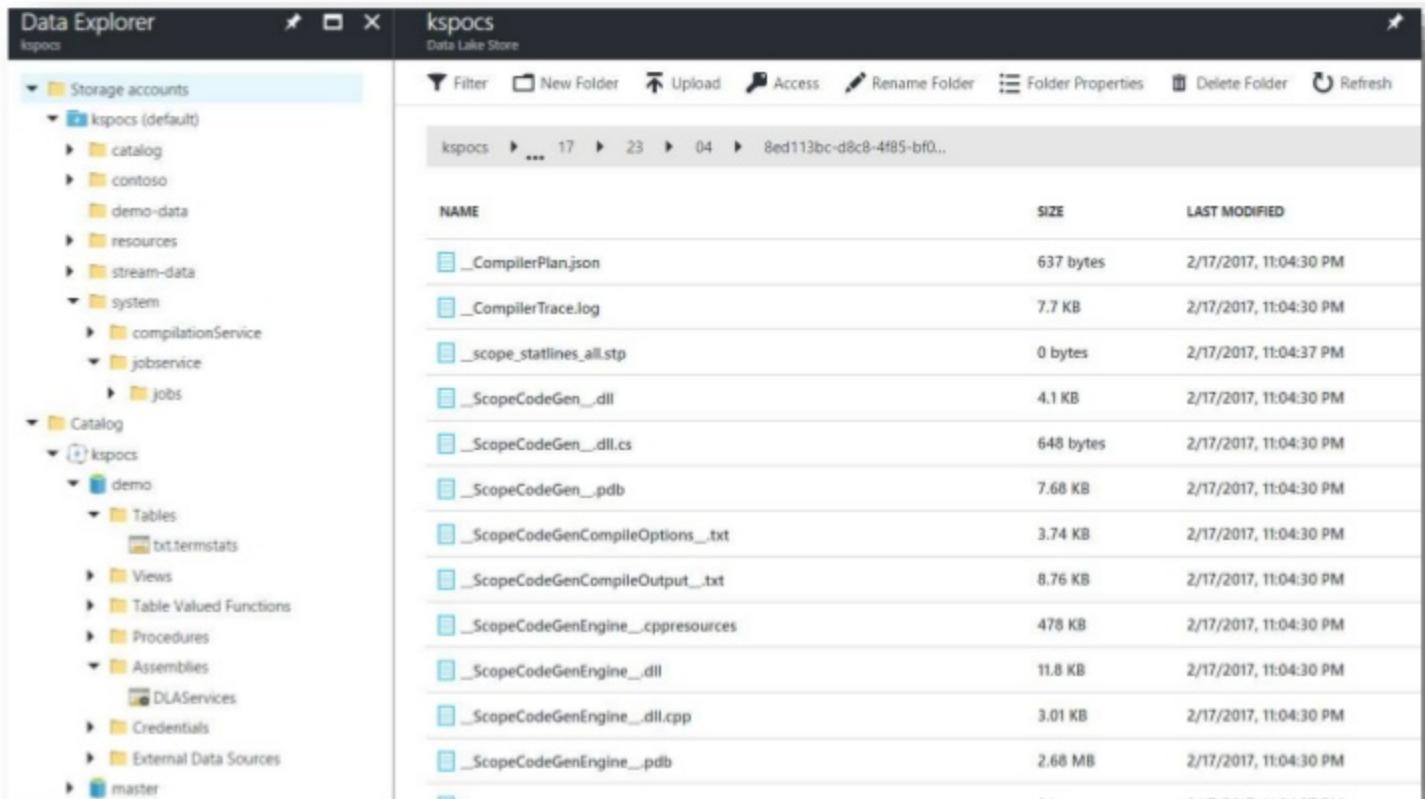
Job Status



Azure U-SQL Jobs

Monitoring Jobs running on Azure

Job files: system/jobservice/jobs/yyyy/MM/dd/jobid/..

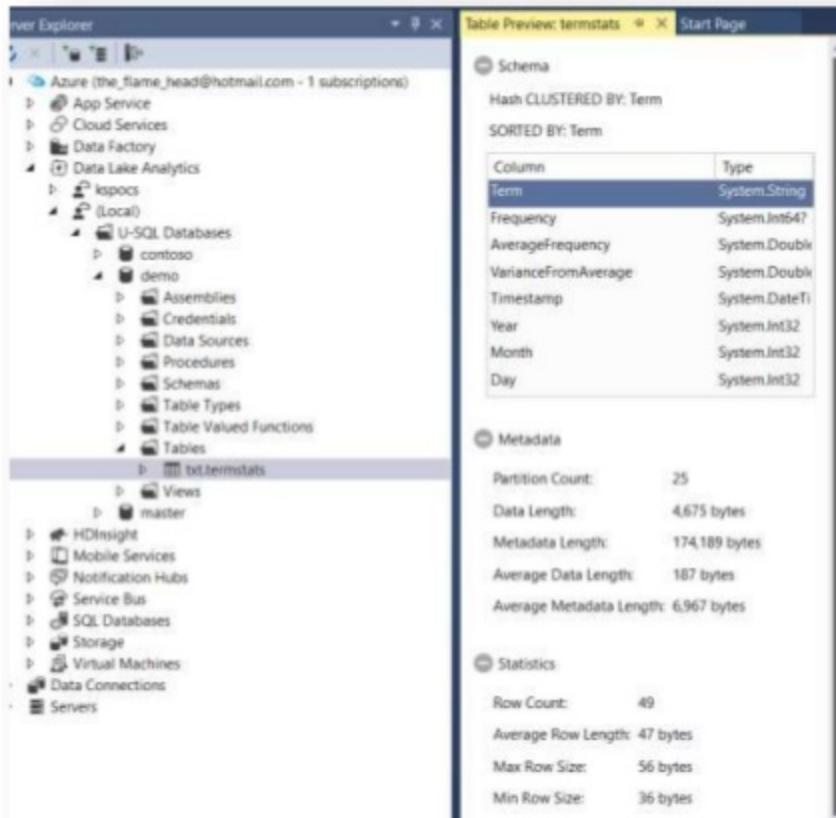


The screenshot shows the Azure Data Explorer interface. On the left, the Data Explorer pane displays a hierarchical view of storage accounts and catalog. Under 'Storage accounts', 'kspocs (default)' is expanded, showing 'catalog', 'contoso', 'demo-data', 'resources', 'stream-data', 'system', 'compilationService', 'jobservice', and 'jobs'. Under 'Catalog', 'kspocs' is expanded, showing 'demo', 'Tables' (with 'txt.termsstats'), 'Views', 'Table Valued Functions', 'Procedures', 'Assemblies' (with 'DLAServices'), 'Credentials', 'External Data Sources', and 'master'. The right pane shows the contents of the 'jobs' folder under 'jobservice'. The path 'kspocs > ... > 17 > 23 > 04 > 8ed113bc-d8c8-4f85-bf0...' is visible at the top. Below is a table listing several files:

NAME	SIZE	LAST MODIFIED
_CompilerPlan.json	637 bytes	2/17/2017, 11:04:30 PM
_CompilerTrace.log	7.7 KB	2/17/2017, 11:04:30 PM
_scope_statlines_all.stp	0 bytes	2/17/2017, 11:04:37 PM
ScopeCodeGen.dll	4.1 KB	2/17/2017, 11:04:30 PM
ScopeCodeGen.dll.cs	648 bytes	2/17/2017, 11:04:30 PM
ScopeCodeGen.pdb	7.68 KB	2/17/2017, 11:04:30 PM
ScopeCodeGenCompileOptions.txt	3.74 KB	2/17/2017, 11:04:30 PM
ScopeCodeGenCompileOutput.txt	8.76 KB	2/17/2017, 11:04:30 PM
ScopeCodeGenEngine.cppresources	478 KB	2/17/2017, 11:04:30 PM
ScopeCodeGenEngine.dll	11.8 KB	2/17/2017, 11:04:30 PM
ScopeCodeGenEngine.dll.cpp	3.01 KB	2/17/2017, 11:04:30 PM
ScopeCodeGenEngine.pdb	2.68 MB	2/17/2017, 11:04:30 PM

Azure U-SQL Jobs

Monitoring Jobs running on Azure



The screenshot shows the Azure Data Explorer interface. On the left, the 'Data Explorer' sidebar lists various Azure services and databases, including 'Azure (the_flame_head@hotmail.com - 1 subscriptions)', 'Data Lake Analytics' (with 'kspocs' and '(Local)' nodes), 'U-SQL Databases' (containing 'contoso' and 'demo' databases with their respective tables like 'Assemblies', 'Credentials', 'Data Sources', 'Procedures', 'Schemas', 'Table Types', 'Table Valued Functions', and 'Tables'), and 'Tables' under 'master'. The 'termstats' table is selected and highlighted.

Table Preview: termstats

Schema

Hash CLUSTERED BY: Term

SORTED BY: Term

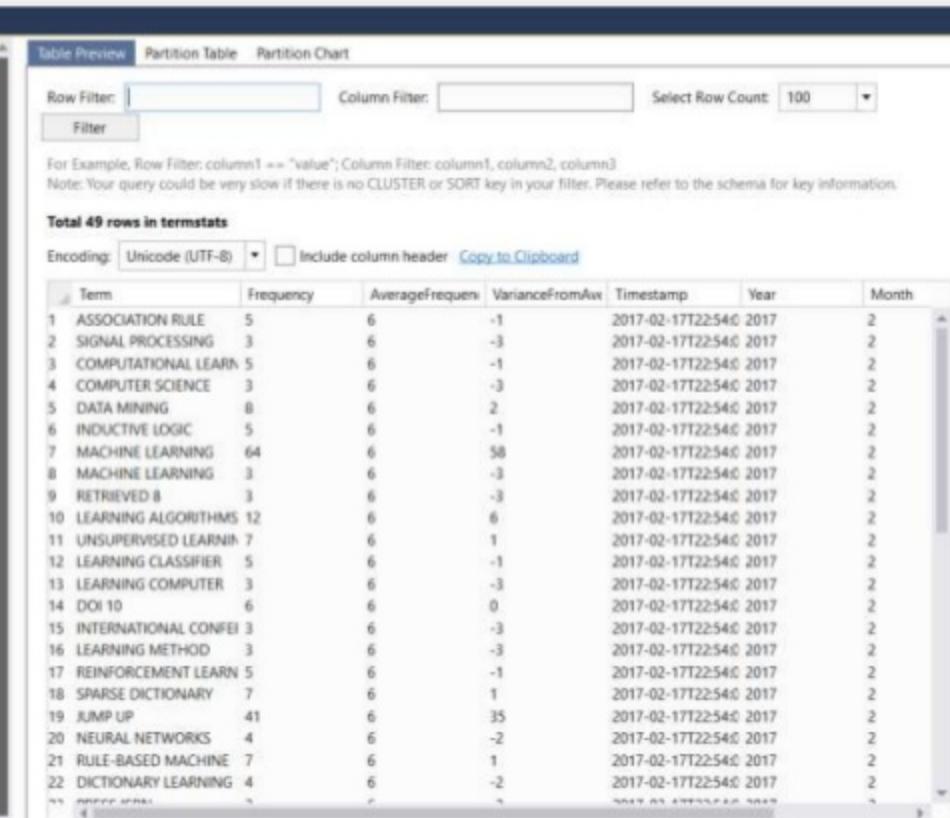
Column	Type
Term	System.String
Frequency	System.Int64?
AverageFrequency	System.Double
VarianceFromAverage	System.Double
Timestamp	System.DateTime
Year	System.Int32
Month	System.Int32
Day	System.Int32

Metadata

- Partition Count: 25
- Data Length: 4,675 bytes
- Metadata Length: 174,189 bytes
- Average Data Length: 187 bytes
- Average Metadata Length: 6,967 bytes

Statistics

- Row Count: 49
- Average Row Length: 47 bytes
- Max Row Size: 56 bytes
- Min Row Size: 36 bytes



The screenshot shows the 'Table Preview' pane for the 'termstats' table. It displays 49 rows of data with the following columns: Term, Frequency, AverageFrequency, VarianceFromAvg, Timestamp, Year, and Month. The data includes various terms and their associated statistics, such as 'ASSOCIATION RULE' with a frequency of 5 and a year of 2017.

Term	Frequency	AverageFrequency	VarianceFromAvg	Timestamp	Year	Month
ASSOCIATION RULE	5	6	-1	2017-02-17T22:54:0	2017	2
SIGNAL PROCESSING	3	6	-3	2017-02-17T22:54:0	2017	2
COMPUTATIONAL LEARN	5	6	-1	2017-02-17T22:54:0	2017	2
COMPUTER SCIENCE	3	6	-3	2017-02-17T22:54:0	2017	2
DATA MINING	8	6	2	2017-02-17T22:54:0	2017	2
INDUCTIVE LOGIC	5	6	-1	2017-02-17T22:54:0	2017	2
MACHINE LEARNING	64	6	58	2017-02-17T22:54:0	2017	2
MACHINE LEARNING	3	6	-3	2017-02-17T22:54:0	2017	2
RETRIEVED	8	6	-3	2017-02-17T22:54:0	2017	2
LEARNING ALGORITHMS	12	6	6	2017-02-17T22:54:0	2017	2
UNSUPERVISED LEARNIN	7	6	1	2017-02-17T22:54:0	2017	2
LEARNING CLASSIFIER	5	6	-1	2017-02-17T22:54:0	2017	2
LEARNING COMPUTER	3	6	-3	2017-02-17T22:54:0	2017	2
DOI 10	6	6	0	2017-02-17T22:54:0	2017	2
INTERNATIONAL CONFERENCE	3	6	-3	2017-02-17T22:54:0	2017	2
LEARNING METHOD	3	6	-3	2017-02-17T22:54:0	2017	2
REINFORCEMENT LEARN	5	6	-1	2017-02-17T22:54:0	2017	2
SPARSE DICTIONARY	7	6	1	2017-02-17T22:54:0	2017	2
JUMP UP	41	6	35	2017-02-17T22:54:0	2017	2
NEURAL NETWORKS	4	6	-2	2017-02-17T22:54:0	2017	2
RULE-BASED MACHINE	7	6	1	2017-02-17T22:54:0	2017	2
DICTIONARY LEARNING	4	6	-2	2017-02-17T22:54:0	2017	2

Azure U-SQL Jobs

Running U-SQL Jobs Using Azure Data Factory

```
{  
  "name": "pipeline-dla-test",  
  "properties": {  
    "activities": [  
      {  
        "type": "DataLakeAnalyticsU-SQL",  
        "typeProperties": {  
          "scriptPath": "pipelines\\test\\process-data.usql",  
          "scriptLinkedService": "StorageLinkedService",  
          "degreeOfParallelism": 3,  
          "priority": 100,  
          "parameters": {  
            "in": "/datalake/input/data-input.tsv",  
            "out": "/datalake/output/data-output.tsv"  
          }  
        },  
      },  
    ],  
  },  
}
```

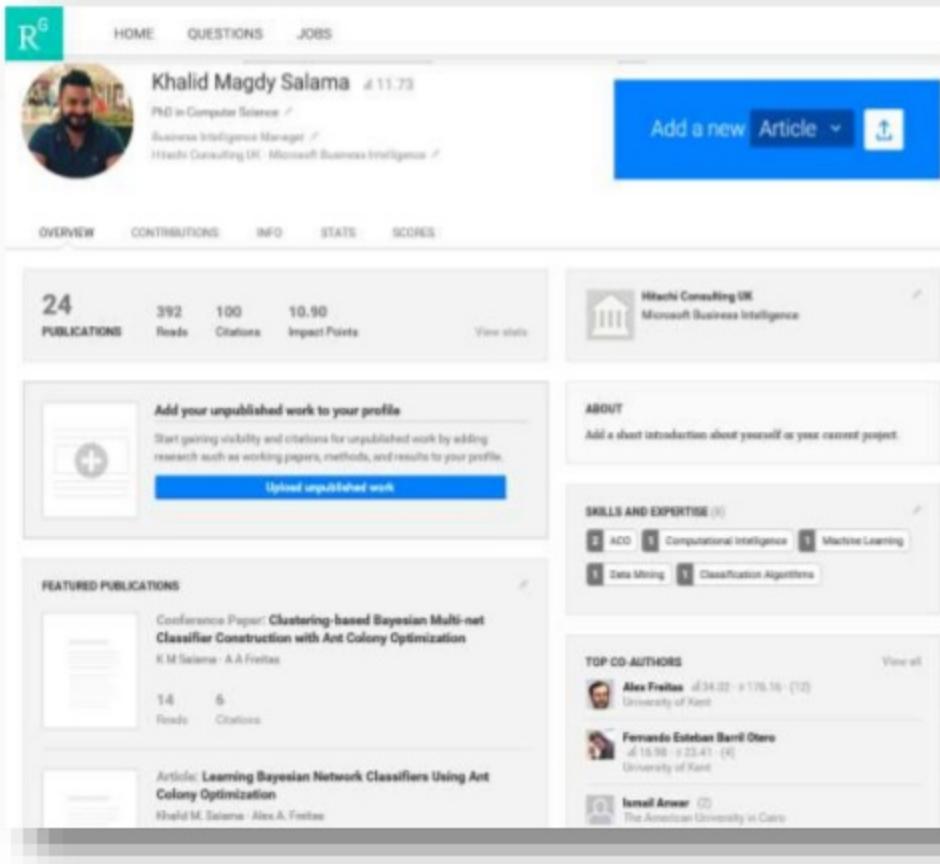


Parameters can be passed to the U-SQL Script to be executed

My Background

Applying Computational Intelligence in Data Mining

- Honorary Research Fellow, School of Computing , University of Kent.
- Ph.D. Computer Science, University of Kent, Canterbury, UK.
- M.Sc. Computer Science , The American University in Cairo, Egypt.
- 25+ published journal and conference papers, focusing on:
 - *classification rules induction,*
 - *decision trees construction,*
 - *Bayesian classification modelling,*
 - *data reduction,*
 - *instance-based learning,*
 - *evolving neural networks,* and
 - *data clustering*
- **Journals:** *Swarm Intelligence, Swarm & Evolutionary Computation, Applied Soft Computing, and Memetic Computing.*
- **Conferences:** ANTS, IEEE CEC, IEEE SIS, EvoBio, ECTA, IEEE WCCI and INNS-BigData.



The screenshot shows Khalid Magdy Salama's profile on ResearchGate. At the top, there is a navigation bar with links for HOME, QUESTIONS, and JOBS. Below the navigation bar is a header section featuring a profile picture of Khalid, his name, and his title as a PhD in Computer Science and Business Intelligence Manager at Hitachi Consulting UK. To the right of the header is a blue button labeled "Add a new Article". Below the header are tabs for OVERVIEW, CONTRIBUTIONS, INFO, STATS, and SCORES. The OVERVIEW tab is selected, showing statistics: 24 PUBLICATIONS, 392 Reads, 100 Citations, and 10.90 Impact Points. There is a "View stats" link next to the impact points. A large central box contains a placeholder for unpublished work with a "Upload unpublished work" button. Below this is a "FEATURED PUBLICATIONS" section displaying two publications by Khalid M. Salama and Alex F. Freitas. The first publication is a Conference Paper titled "Clustering-based Bayesian Multi-net Classifier Construction with Ant Colony Optimization" with 14 reads and 6 citations. The second publication is an Article titled "Learning Bayesian Network Classifiers Using Ant Colony Optimization" with 14 reads and 6 citations. To the right of the profile page are several sidebar boxes: "Hitachi Consulting UK Microsoft Business Intelligence", "ABOUT" (with a placeholder for a short introduction), "SKILLS AND EXPERTISE" (listing ACO, Computational Intelligence, Machine Learning, Data Mining, and Classification Algorithms), and "TOP CO-AUTHORS" (listing Alex Freitas, Fernando Esteban Barril Otero, and Ismail Awad).



Thank you!