Udacity: Data Engineering with Azure

Data Modeling

1. Introduction to Data Modeling

Data modeling - Relational Database Management System (RDMS)
- PostgreSQL
- mySQL

- NoSQL
  - Cassandra
  - MongoDB
  - denormalized
    queries first - 1 table per query
    can't join tables

OLAP vs OLTP

Normalized vs Denormalized tables
- 3NF
- fast read
  slow write

Fact tables, Dimension tables
- star
- snowflake

Cloud Data Warehouse with Azure

1. Introduction to Cloud Data Warehouses with Azure

Data Sources →	Data Engineering →	Data Consumers
- CSV - video, audio	- data pipelines	- analytics
- JSON	• ingest	- visualization
- database	• transform	- data science
- logs, system data, sensors, devices,	• store	

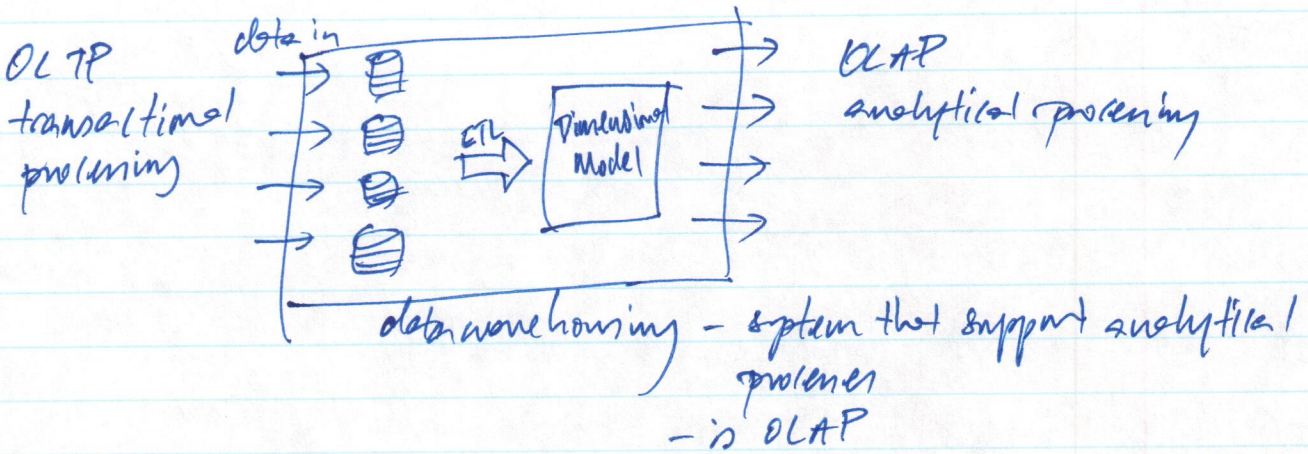
2. Introduction to Data Warehouses

In Business:

Operational Processes - customers, inventory, delivery

vs.

Analytical Processes - HR, marketing, management.



If same database for OLAP and OLTP - database scheme too complex for business analysts
- analytic queries require too many joins
- o.k. if database is small.

Data Warehouse:

- Kimball
  - is a copy of transaction data specifically structured for query + analysis

- Inmon
  - is subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions.

- Reinardi:
  - is system that retrieves + consolidates data periodically from source into a dimensional or normalized data store. Keeps years of history. Query for business intelligence or other analytical activities. Updated in batches.

Dimensional Modeling:
3NF - expensive joins
- hard to explain to business users

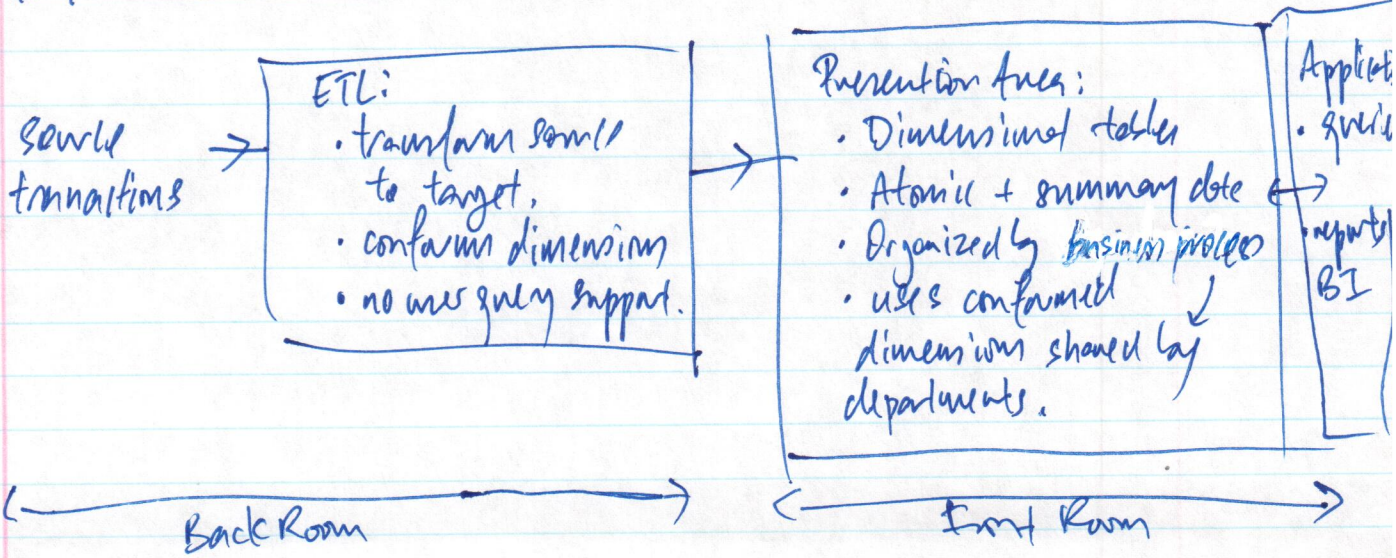
Star schema - joins with dimensions only
- good for OLAP, not OLTP
- fast query

Facts table
- quantifiable business event or metrics

Dimension table
- context of business event - e.g. who, what, why, where, etc.

3NF to star schema - ETL
- extract - join tables
  - change types
  - add new columns
  } transform
- load
  - insert into facts, dimension tables.

Types data warehouses:
1. Kimball's Bus Architecture

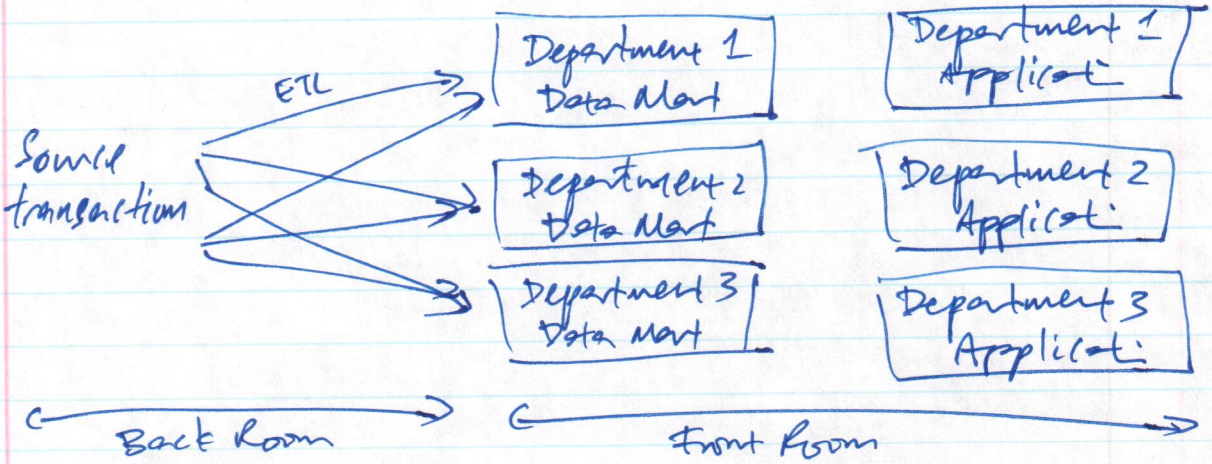


Extract
- transfer data to warehouse.

Transforming
- integrate many sources together.
- cleansing - duplicates, inconsistencies, missing values.
- produce diagnostic metadata.

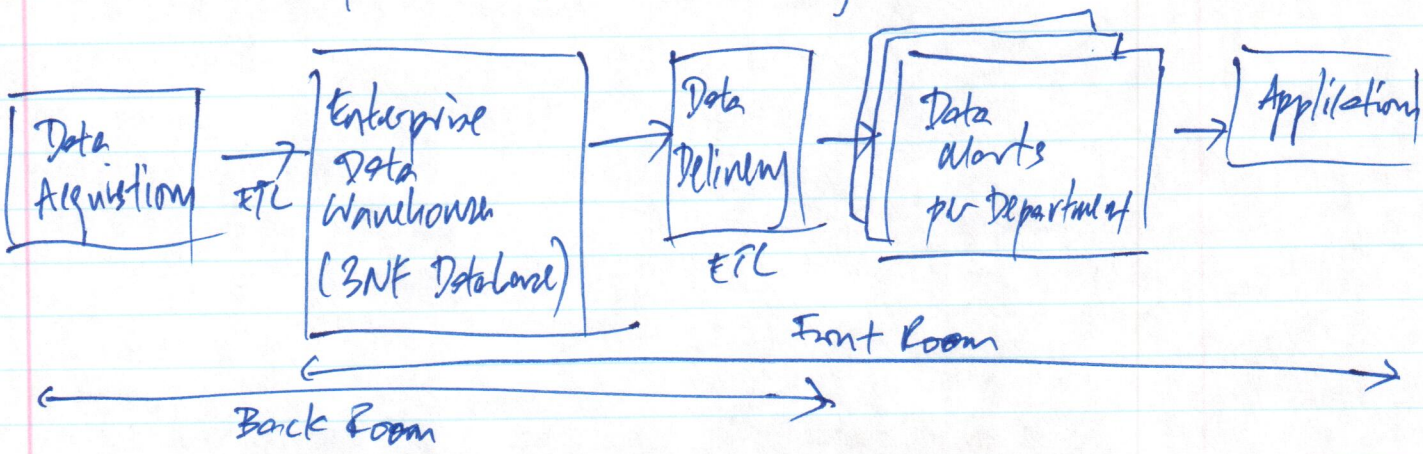
Loading
- structuring + loading data into dimensional data model.

2. Independent Data Marts



- Separate ETL process + dimensional models called data marts.
- different fact tables for same events, no conformed dimensions.
- generally discouraged !
- can lead to inconsistent views.

3. Inmon's Corporate Information Factory (CIF)

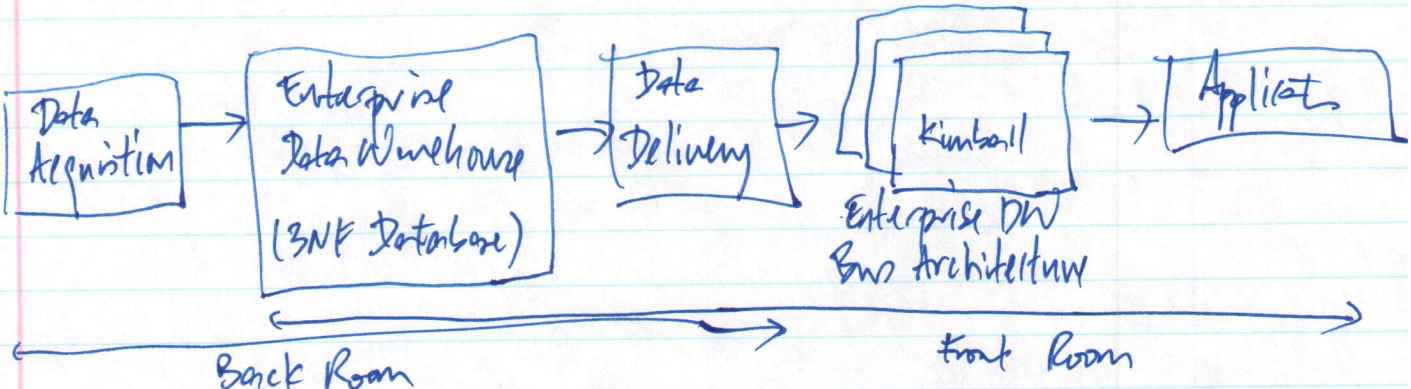


- 2 ETL processes
  1. source → 3NF database
  2. 3NF database → department data marts

- 3NF database acts as enterprise-wide data store
  - single integrated source of truth for data marts
  - could be accessed by end-users if needed

- data marts are dimensionally modeled, mostly aggregated, unlike Kimball

4. Hybrid Kimball Bus + Inmon CIF

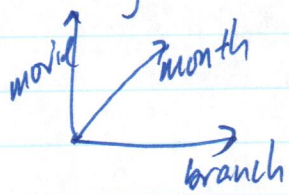


- Maintain 3NF Database for ground truth.
- Remove data mart
- Replace with enterprise data warehouse (Kimball - conformed dimension).

OLAP Cubes

- After creating facts + dimensions tables in star schema, create OLAP cubes.

e.g.



- OLAP cube is aggregate of number of dimensions
- roll up, drill-down, slice + dice.

- Roll up - sum up colums.
- Drill down - decompose into smaller grains.
- Slice - reduce N dimensions to N - 1 where N^{th} dim is constant value.
- Dice - compute sub-cube - restrict range of values in columns, while retains some N dimensions.

- GROUP BY CUBE (x, y, z) will make one pass through facts table, and aggregate all possible combinations of groupings of lengths 0, 1, 3

e.g. total revenue, revenue by x, revenue by x, y, revenue by x, y, z
" by y, y, z,
" by z, x, z,

- Alien OLAP cubes
  1. Pre-aggregate OLAP cubes, save on non-relational database (MOLAP) old approach.
  2. compute cube on the fly, ROLAP

3. ELT and Data Warehouse Technology in Cloud

Benefits of cloud DW over on-premise,

1. Scalability
   - expand with more data.

2. Flexibility
   - add & remove services from pipeline as needed.
   - change to accomodate business needs changes.

3. cost shifting
   - perform costly + time consuming transformation of data as last part of process for 'just-in-time' business prioritization.

ETL vs ELT
- ETL - transformation happens on intermediate server
- ELT - " " " destination server via staging tables.
  - better scalability
    " flexibility
    " cost shifting
    " performance on large datasets
  more flexibility for NoSQL datasets

Relational Database:
- oracle
- Microsoft SQL server
- postgreSQL
- mySQL
- maria DB

Microsoft Azure:
- Azure SQL Database
- " Database for MySQL
- " " MariaDB
- " " PostgreSQL

GCP -
- cloud SQL (mySQL, postgreSQL, MS SQL server)

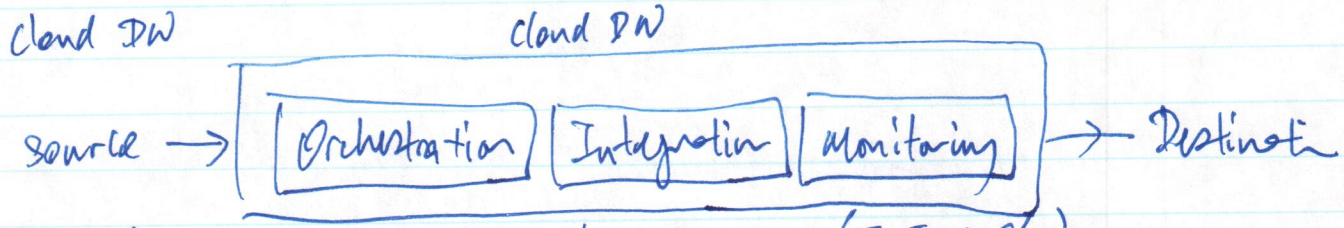
AWS
- Amazon RDS (mySQL, postgreSQL, maria DB, oracle, MS SQL)

NoSQL database:
- key value
- document
- column oriented
- graph
- time series

Azure
- cosmoDB
- Gremlin - graph database
- mongoDB - document
- Cassandra - column oriented

GCP
- big table - column oriented
- firestore - document
- mongoDB atlas - document

AWS
- DynamoDB - key value
- Document DB - document
- Keyspaces - column oriented
- Neptune - graph
- Time stream - time series



ETL/ELT
- Azure Data Factory
- AWS Glue
- GCP Dataflow

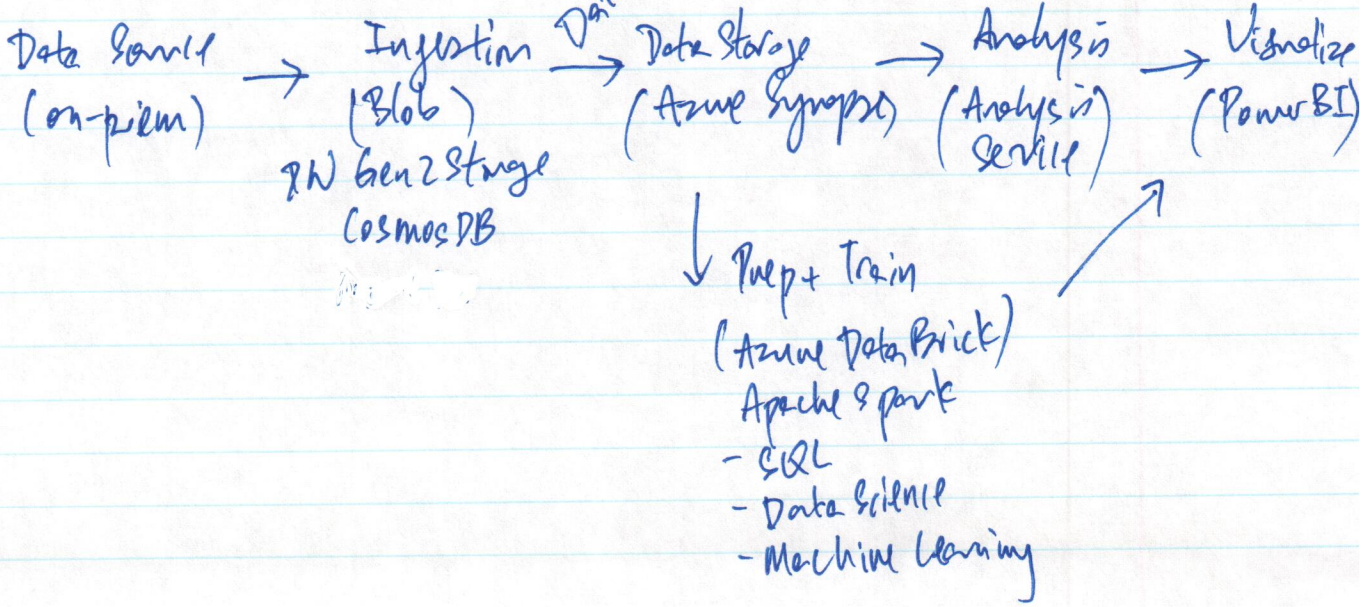
Streaming Data (IoT, web)
- Azure - Streaming Analytics
- AWS - Kinesis
- GCP - DataFlow

3rd party companies providing ETL/ELT
- informatica
- talend
- xplenty
- matillion

Cloud DW Solutions
- Azure Synapse
- AWS Redshift
- GCP Big Query

4. Azure Data Warehouse Technologies

Azure DW



Azure Portal → Synapse Analytics workspace → Open Synapse Studio

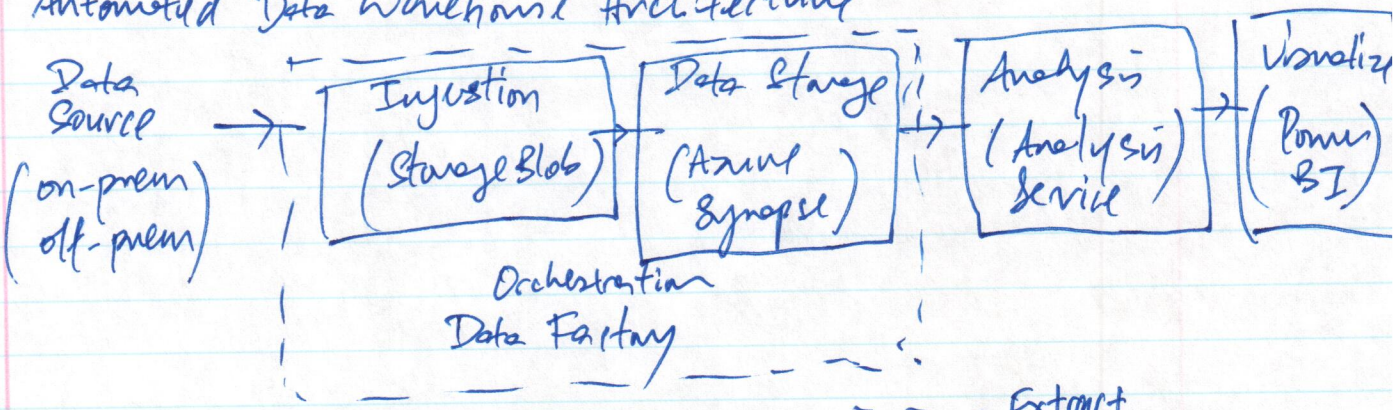
Data - add databases
Develop - Add notebooks to query data. e.g. SQL scripts.
Integrate - Add pipelines from data sources
Monitor - look at runs for each of above.
Manage - manage components Synapse

Cloud Data Storage:
- Azure Data Warehouse Gen 2 - traditional DW architecture
- Azure Dedicated SQL Pools - relational data storage
- Blob storage - file based storage
- CosmosDB for No SQL - column-oriented, a document db.

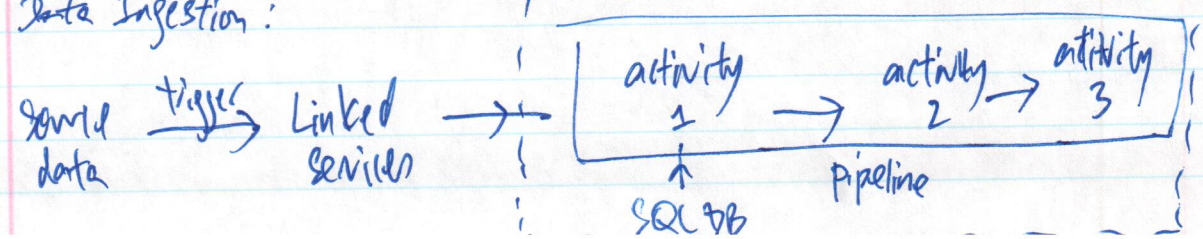
ETL/ELT Pipelines:
- Azure Data Factory - data integration, data flows
- Azure Databricks - use Spark to create ETL pipelines (Python, Java, SQL)
- Azure Polybase - use TSQL to query blob storage to support ETL

Implementing Data Warehouses in Azure

Automated Data Warehouse Architecture



Data Ingestion:



Create Linked Services:

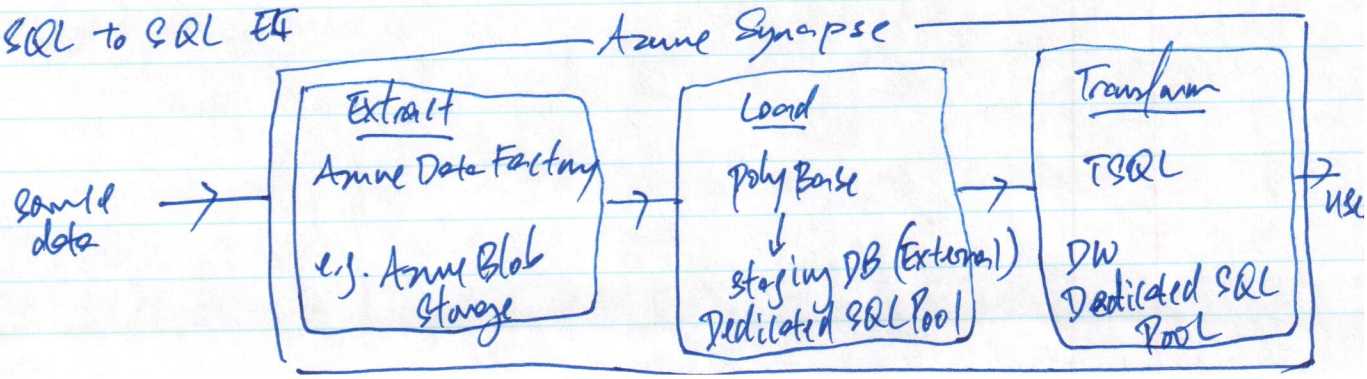
Azure Portal → Azure Synapse Workspace → Open Synapse Studio

↳ managed - external connection
└ linked services
└ + New
└ select type
└ Name
└ From Azure Subscripti
or manually
username / password
Test Connection ←

Ingest - form source to destination
- Built in copy task
- Run Once / Schedule / Tumbling Window
- source type
Connection
└ source tables
└ Preview
└ Target type
Connection
└ destinati
file type

→ View in Data

SQL to SQL ETL



Create Staging Tables:

Synapse Studio → manage → Analytics pool → SQL pools → dedicated

Data → Workspace → Tables → External tables (staging tables)

→ Linked → Azure Datalake Storage Gen 2 → Blob Storage → New SQL Script → Select TOP 100 rows

Create external table
Bulk load

File type
Select SQL pool
Select database
External table name
Create external table
└ Automatic
└ Use SQL script

Open script
Edit
Run

view to check
DROP table
CREATE table

Create Table:

Develop → + Add New Resource → SQL Script → connect to dedicated SQL pool

```
CREATE TABLE newTables (
    id      VARCHAR(50)
    date    DATE
)
GO;
```

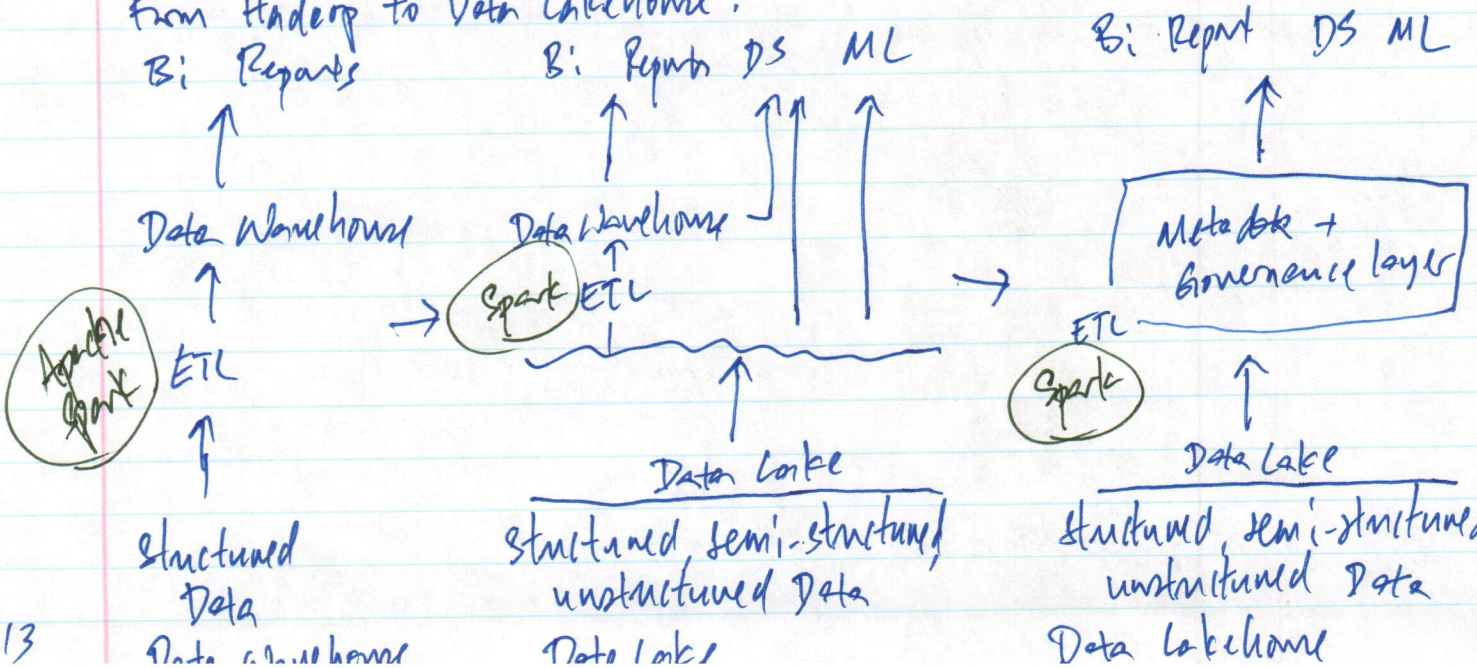
```
INSERT INTO newTables (id, date)
SELECT id, date
FROM staging_table

GO;
```

6. Big Data Ecosystem, Data Lakes + Spark

		Lakehouse
Data Analysis		BI Data Science Machine Learning
Massive Parallel Processing	MapReduce	Spark
Distributed File Storage	Hadoop HDFS	Data Lake

- HDFS - distributed storage with high-throughput access to data.
- MapReduce - massive parallel processing for large amount of data.
- → Apache Spark - multi-programming APIs for processing data
  - interface for iteratively developing data interactive engineering + data science solutions.
- → Data Lake - process large amount of structured + unstructured data.
- → Lakehouse - data lake + data warehouses.

From Hadoop to Data Lakehouse:



Hadoop
~~HDFS~~ - ecosystem of tools for big data storage + analysis.
- writes intermediate results to disk whereas Spark keeps data in memory for as long as possible.
- Spark is faster; does not include file storage system.

Hadoop MapReduce - processing + analyzing large datasets in parallel.

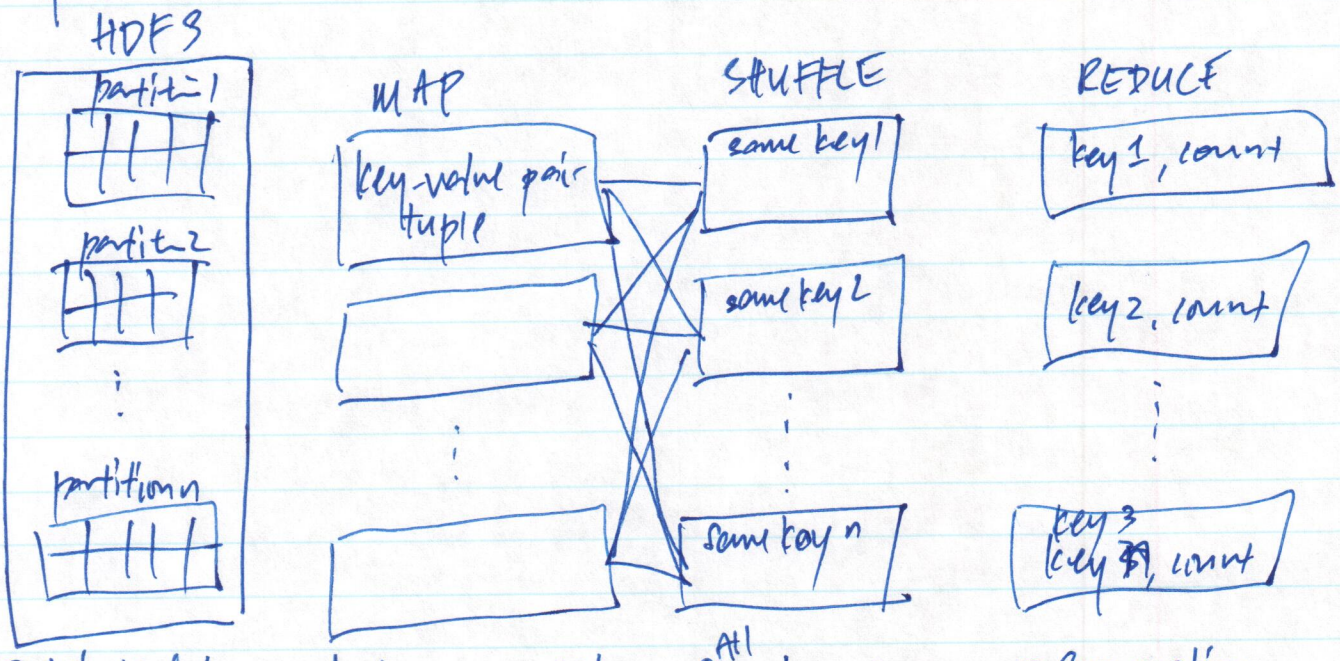
Hadoop YARN - resource manager to schedule jobs across cluster. what resources are available, assigns tasks.

HDFS - big data file storage system that splits data into chunks + stores chunks across cluster of computers.

Apache Pig - SQL-like language that runs on top of Hadoop MapReduce.

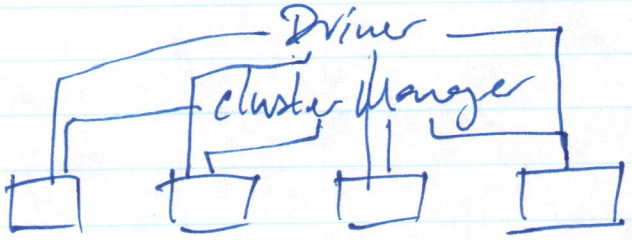
Apache Hive - another like Apache Pig

MapReduce:



- Distribute data across cluster
- Analyze + convert to key-value pair
- All same keys on same cluster
- Summation

Spark:



→ monitors available resources + make sure all machines are responsive

cluster Modes:

- Standalone
- YARN (from Hadoop)
- MESOS (from UCBerkeley)

} 3 types of cluster managers

Use cases for Spark:
- data analytics
- machine learning
- streaming
- graph analytics

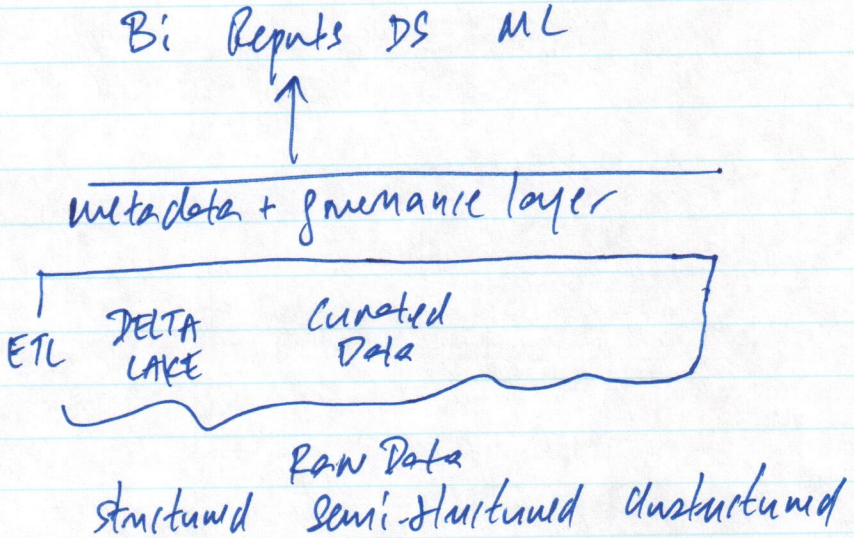
Limitations Spark:
- Spark streaming has latency of at least 500 ms since it operates on micro-batches of data, rather than 1 record at a time.
- Other streaming tools - Storm, Apex, Flink can reduce this latency.
- Can't do ML with Spark as it only supports algorithms that scales linearly with data.

Key feature Data Lake: and semi-structured.
- includes unstructure data, in addition to structured.
- lower costs with ETL/ELT operations using by data tools.
- schema-on-read over schema-on-write, so lowers costs + work on ingesting large amounts of data.

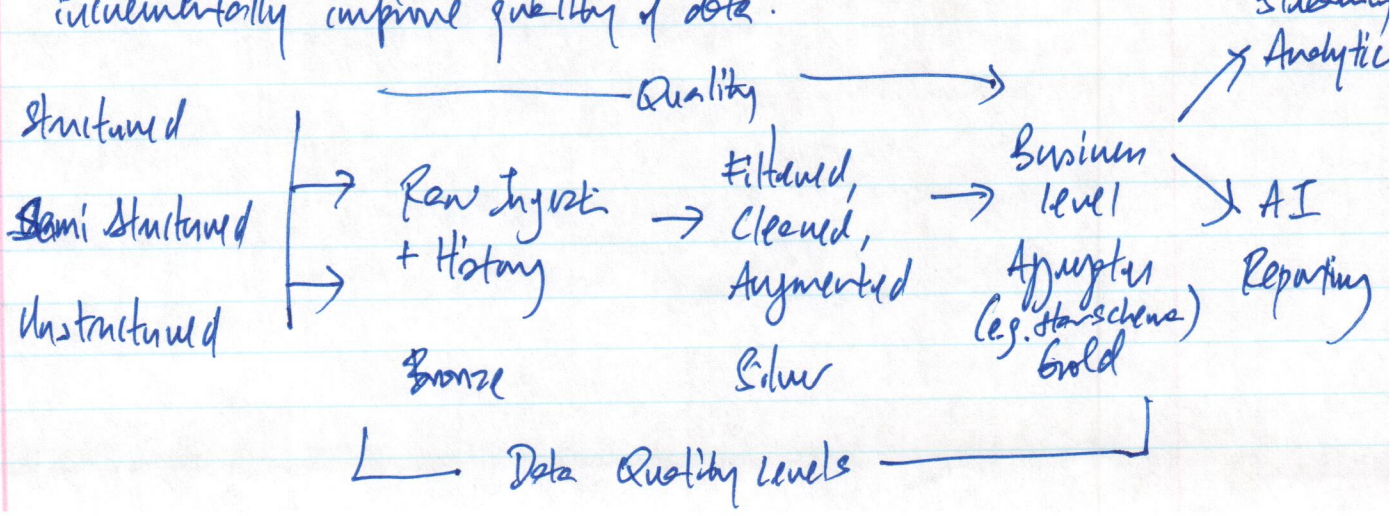
Apache Spark used on
- DE
- Data Analytics
- DS

Data Lake architecture limitations:
- lost atomic transactions
  - failed jobs left in corrupt state.
- lost quality enforcement
  - inconsistent + therefore unusable data.
- lost consistency in data structures
  - impossible to stream + batch process while ingesting.

Data Lakehouse
- add it of metadata + governance layer on top of data lake.
  - center pool of raw data + curated data!



- ingest large amounts of data quickly
  incrementally improve quality of data.



7. Data Wrangling with Spark

Functional programming - Spark
vs
Procedural programming - C++ / Python

Pure functions - functions preserve their inputs + avoid side effects.

Spark - lazy evaluat - build steps by steps directions
└ Directed Acyclic Graph (DAGs)
- wait till necessary, then run DAGs
- multi-step combos are called stages.

```
import findspark
findspark.init('spark-...-bin-hadoop')

import pyspark
sc = pyspark.SparkContext(appName='example')

distributed_list = sc.parallelize(list)
distributed_list.map(lambda x: x.lower()).collect()
                                          ↳ gather results from all machines.
distributed_list.collect()
    ↳ not mutated
```

Hadoop + HDFS
- splits files into 64 or 128 mb blocks. for digestible chunks.
- store in fault tolerant way.

Spark Dataframe:

```
import pyspark
from pyspark import SparkContext, SparkConf
configure = SparkConf().setAppName('name').setMaster('ip address' or 'local')
sc = SparkContext(conf = configure)
from pyspark.sql import SparkSession
spark = SparkSession.builder \
                    .appName('example') \
                    .getOrCreate()

spark.sparkContext.getConf().getAll()

log = spark.read.json(path_file)
log.printSchema()
   .describe()
   .show(n=1)  <-
   .write.save(out_path, 'csv', header=True)
```

```
spark_df to pandas_df:
pandas_df = spark_df.toPandas()
```

Typical spark functions:
- select()
- filter()
- where()
- groupBy()
- sort()
- dropDuplicates()
- withColumn()

Aggregate functions:
- count()
- countDistinct()
- avg()
- max(), min()
- agg() after groupBy()

User Defined Function (UDF)

Window functions
- partitionBy
- rangeBetween, rowsBetween.

Spark SQL:

```
df.createOrReplaceTempView('table')

spark.sql("SELECT * FROM table LIMIT 5').show()

sql_df = spark.sql('''
            SELECT *
            FROM table
            LIMIT 5
            ''').collect()

pandas_df = sql_df.toPandas()
```

UDF:

```
spark.udf.register('get_x', lambda x: x)
spark.sql('''
            SELECT *, get_x(x) AS x
            FROM table
            LIMIT 1
            ''').collect()
```

Resilient Distributed Data Set (RDD)

Spark version — 1.3 DataFrame API
- <1.3 RDD — low level abstraction of data
- 2.0 DataFrame Dataset API

Spark Accumulators ↖ 8. Spark Debugging and Optimizati

```
x = SparkContext.accumulator(0, 0)
```

Spark Broadcast

Action Functions
- trigger lazy evaluation functions
- e.g. .write()

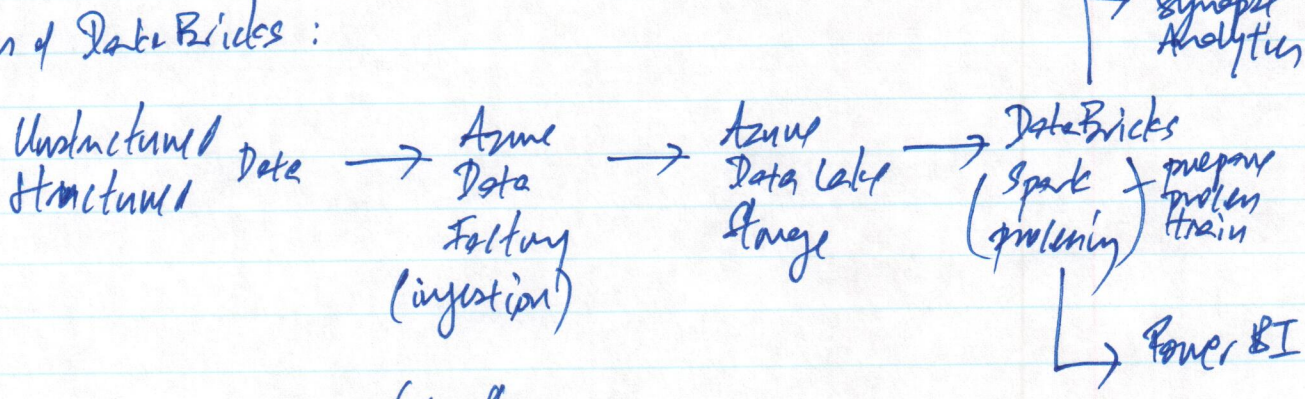
Data Skewness
- check MIN, MAX, RANGES, value_counts()
- change workload division - create composite keys
- break data into smaller partitions.
  - ↳ df.repartition(number_of_workers)

Q. Azure Data Bricks

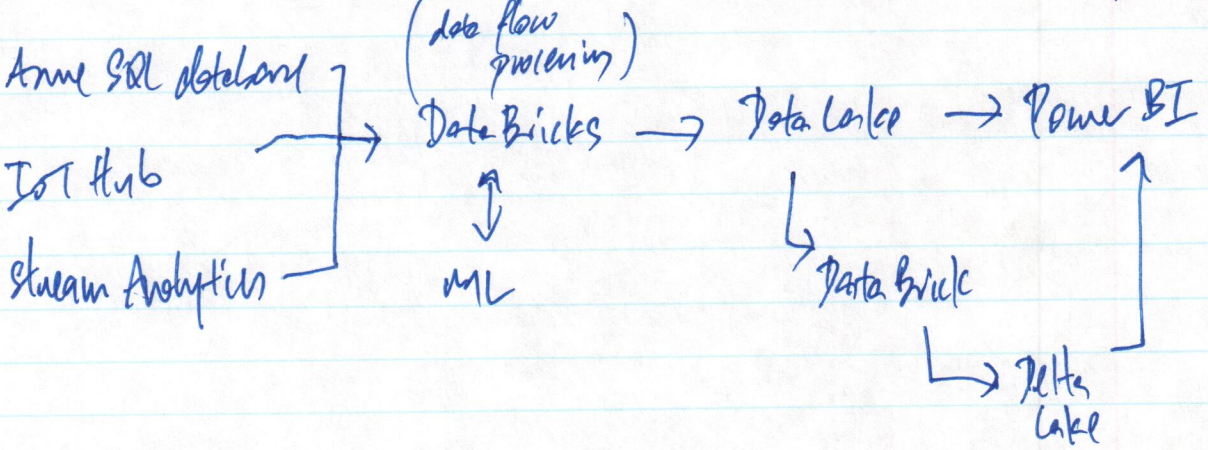
Azure Databricks contains:
- distributed filesystem (storage)
- workspace (data engineering, data analytics, data science)
  (ingesting, profiling, transforming, storing, serving data
- Spark clusters
  (Spark API, Spark SQL, Spark Data Frames

Users of Data Bricks:

1.



2.



Azure Portal → Create Resource → Azure DataBricks
↳ Name
Resource grp
Region
Pricing - standard
Launch Workspace
↳ Compute → Create Cluster
↳ Name
Cluster Mode
DataBricks Runtime
Autopilot options
Worktype
↳ Min Workers, max workers → delta minimum cost
Driver Type
→ Create Cluster
→ Stop / start / delete

→ ~~Create~~ Workspace → Shared
↳ Create
↳ Notebook
↳ Name
Language
Cluster

cluster must run first.

Write To Delta Lake:

```
df = spark.read.format('csv').option('sep',';').option('header','True').load('filepath')
display(df)

df.write.format('delta').mode('overwrite').save('/delta/filepath')
                                            .saveAsTable('dataTable')
```

↳ Create Table

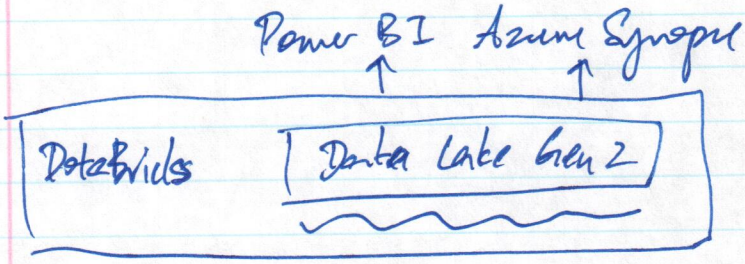
Data → Database Tables → default → Table
↳ datatable

→ DBFS → /delta
↳ data → _delta_log
part..... parquet

Settings → Workspace Settings
↳ Admin Console → ↳ Advanced
↳ DBFS File Browser - enabled.
(Refresh browser)

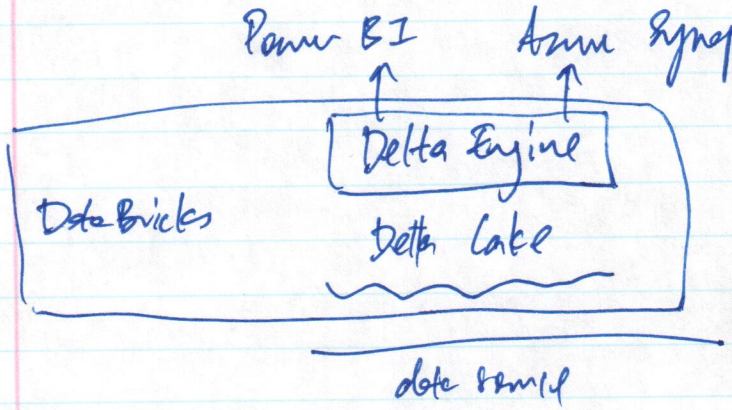
10. Data Lakes and Lakehouse with Azure DataBricks

Data Lake Gen 2:

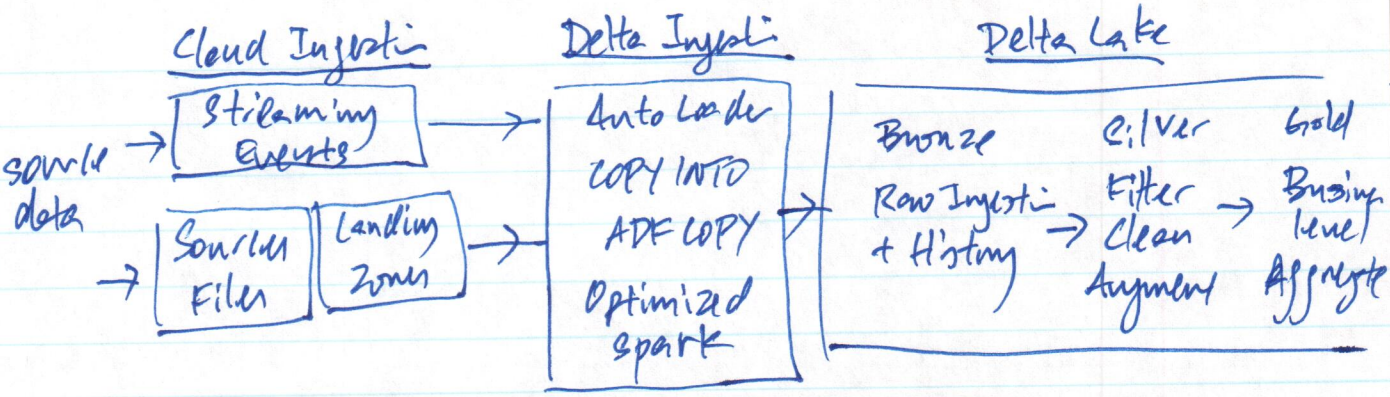


- adds Azure Blob Storage
- Hierarchial namespaces to enable better organization of information.
- accessible using Hadoop APIs
- integrates with Azure Synapse, Data Factory, Databricks.

Delta Lake:



tasks:
1. create workspace
2. ingest data
3. create, update, delete Tables
4. read, write Data



Create Table:

```
spark.sql ('CREATE TABLE table (' \
            'loan_id BIGINT, ' + \
            'paid_amnt DOUBLE' )
```

or)

```
df.write.format ('delta') \
  .mode ('overwite') \
  .saveAsTable ('table')
```

or,

```
Spark.sql ("CREATE TABLE datatable USING DELTA \
            LOCATION '/delta/data'" )
```

Drop Table:

```
spark.sql ("DROP TABLE IF EXISTS datatable")
```

Read data:

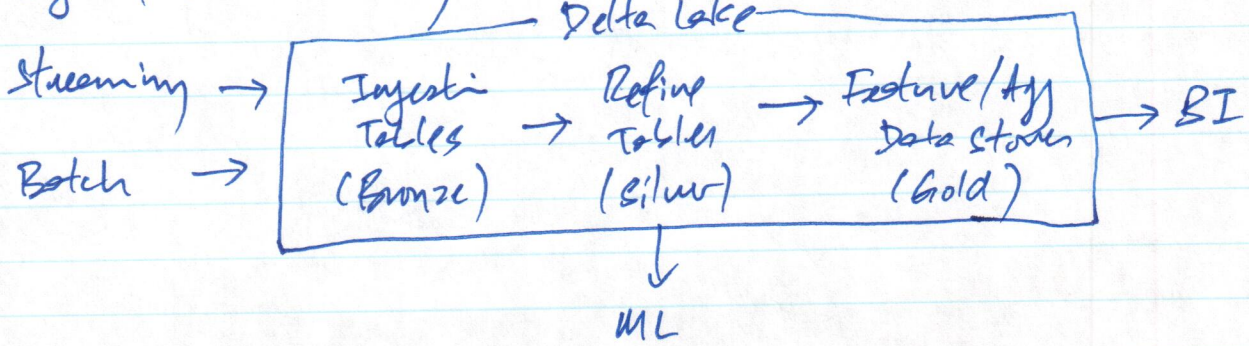
```
spark.table ('default.tablename')
spark.read.format ('default').load ('/delta/tablename')
```

Write data:

```
df.write.format ('delta').mode ('append') \
  .saveAsTable ('default.tablename')
  .save ('/delta/tablename')
```

```
spark.sql("CREATE TABLE tablename " \
          "USING DELTA LOCATION '/delta/data' ")
```

Stages of Data Refining:



Ingestion:

```
df = spark.read.format('csv') \
        .option('inferSchema', 'false') \
        .option('header', 'true') \
        .option('sep', ',') \
        .load('/FileStore/demo2/file.csv')

df.write.format('delta') \
        .save('/delta/bronze_file')
```

Refine:

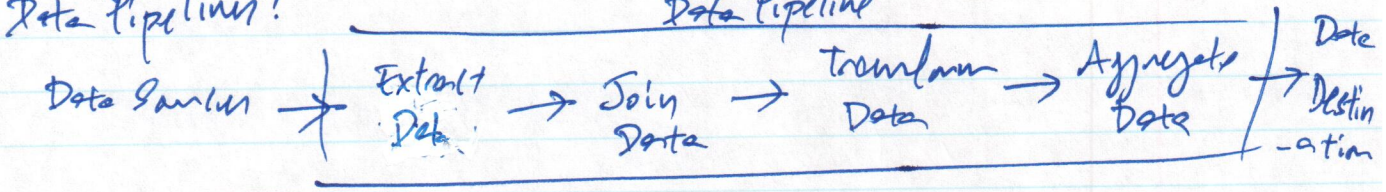
```
df = spark.read.format('delta') \
        .load('/delta/bronze_file')

from pyspark.sql.functions import sha2, concat_ws
df = df.dropDuplicates(df.columns)
df = df.withColumn('hash', sha2(concat_ws('||', *df.columns), 256))

df.write.format('delta') \
        .mode('overwrite') \
        .saveAsTable('silver_file')
```

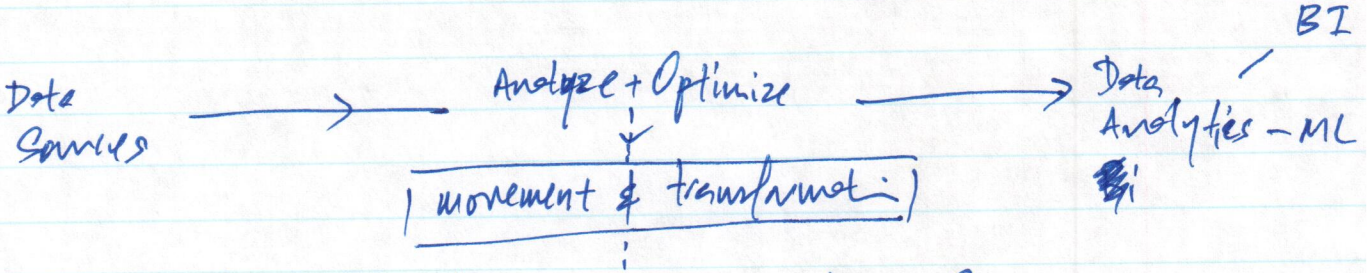
Aggregate:

```
df = spark.table('silver_file')

df = df.groupBy('x', 'y').count()

df.write.format('delta') \
    .mode('overwrite') \
    .saveAsTable('gold_file')
```

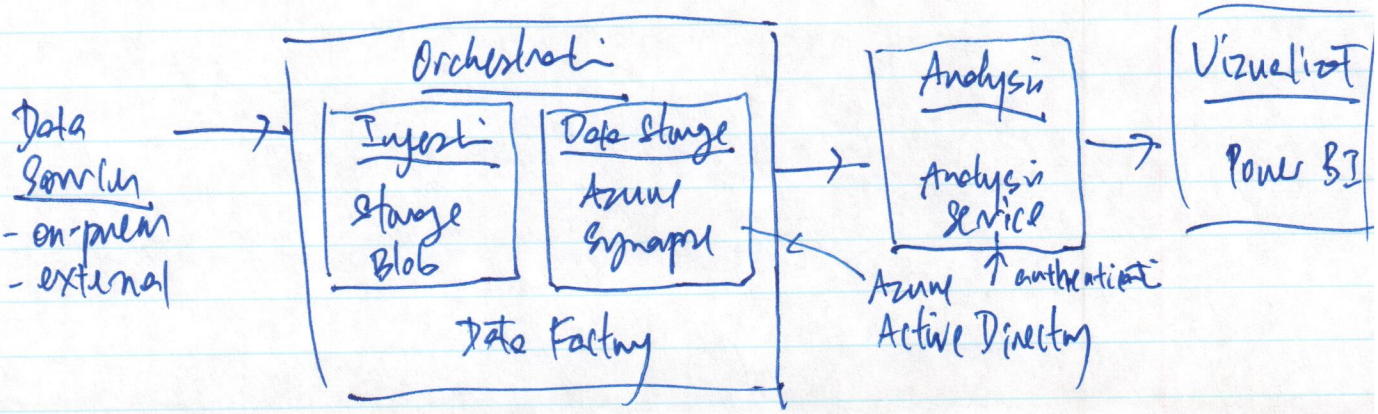
11. Data Pipelines with Azure

Data Pipelines:

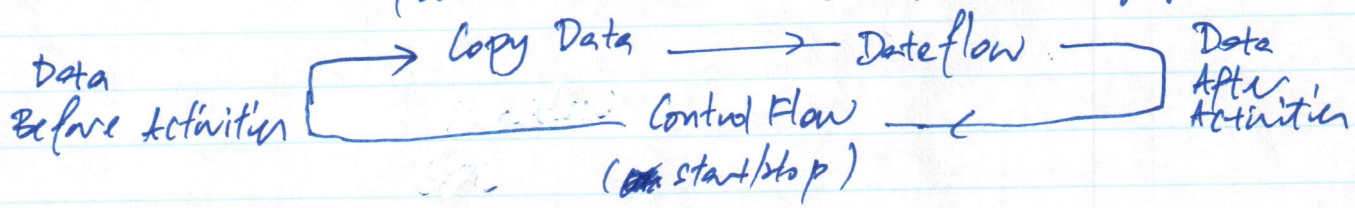




- provide cloud-based code-free ETL or ELT as a service to orchestrate data movement



Pipeline Activities:



Azure Portal → Create Azure SQL Database
↳ Query Editor
└ login
└ query

→ Create Azure Synapse
↳ Open Synapse Studio
↳ Manage - SQL pools - New
↳ (create)

↳ Develop - New SQL script
└ create tables in SQL pools

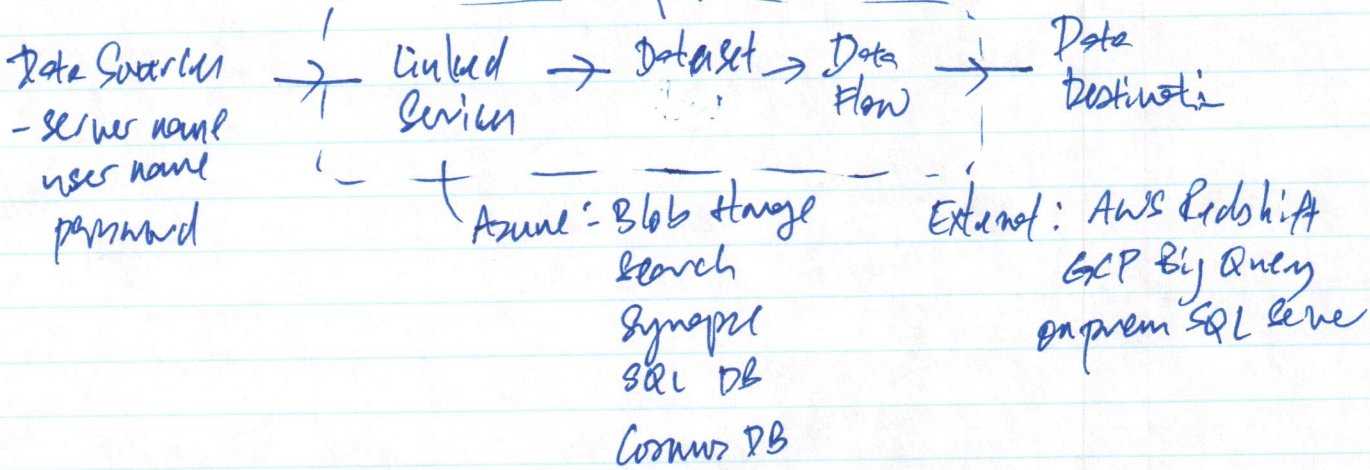
→ Create Azure Data Factory
↳ Name
Region
Version
↳ Git (configure later)
↳ Create
↳ Open Azure Data Factory Studio

→ Create Storage Account
↳ Advanced
↳ Enable hierarchical namespace
↳ Create
↳ Storage Browser → Blob containers - + add container

- view data source objects
not make copy.

Linked Services:

Data | Pipeline



File: AWS S3
Google cloud
FTP

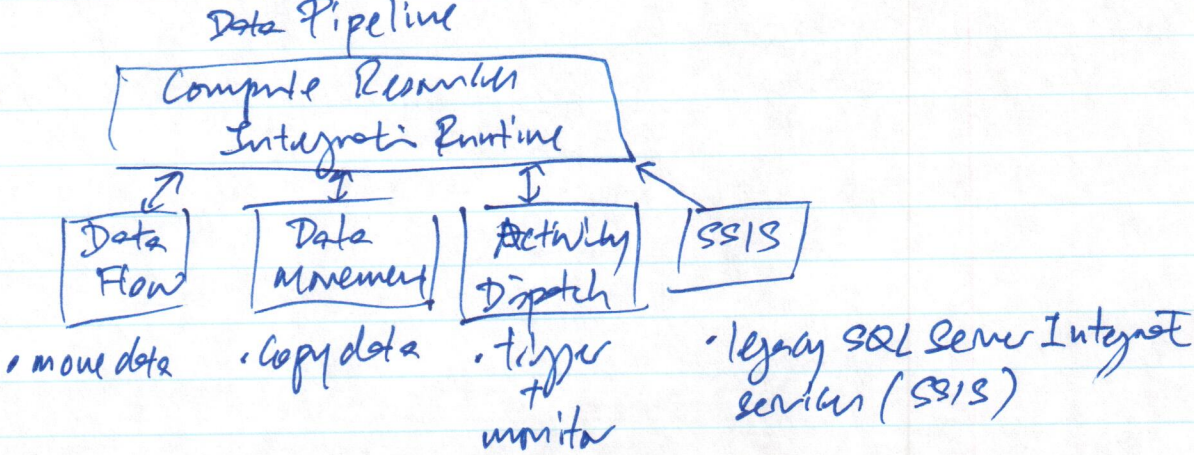
NoSQL: MongoDB
Cassandra
....

Azure Data Factory
↳ manage - Linked services - SQL (white list IP address)
- Synapse
- storage account

↳ Author - Datasets - New Datasets
└ SQL
└ name
linked service
table name
└ Azure Synapse Analytics
└ name
linked service
table name

- Publish

Integration Runtime (IR):



3 types:

1. Azure IR
   - data flow } between cloud stores
   - data movement }
   - dispatch to external compute
   - SSIS in public or private network
   - serverless compute.

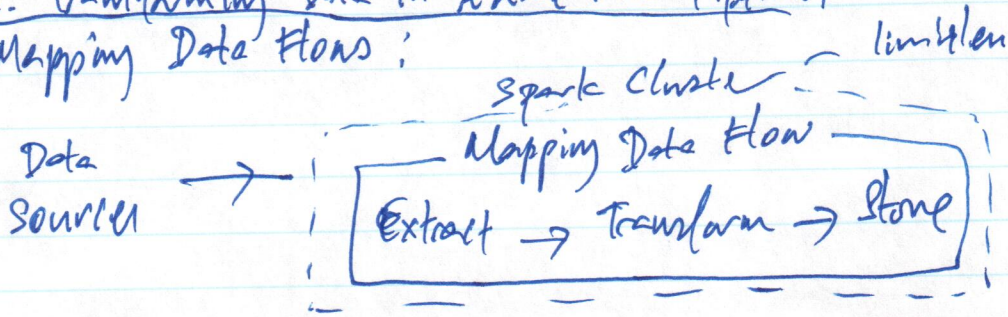
2. Self-hosted IR
   - data movement between cloud stores and private network.
   - dispatch to external compute in Azure Virtual Network.

3. Azure SSIS IR
   - lift + shift existing SSIS to execute in Azure.

Azure Data Factory
↳ manage → Integration Runtimes → + New
↳ Azure, Self-hosted
↳ Azure
↳ name, region
↳ Time to Live
Compute type, Core

12. Transforming Data in Azure Data Pipelines

Mapping Data Flows:



3 types:

1. Schema modifiers
   - create new derived columns based on calculations, aggregate data, or pivoting data

2. Row modifiers
   - change rows e.g. filtering, sort, alter, based on insert/update/delete/upsert policies.

3. Multiple inputs/outputs
   - generate new data with a join, union, or splitting data.

- flowlets - prevent logic duplications + apply consist logic
- formatters - change one format to another

Data flows
- integrated with Visual Expression Builder in Azure Data Factory to perform logic transformations as simple expression.

- Expressions - column mapping
  - functions
  - parameters
  } evaluated Spark data types at runtime.

+ - input schema - reference columns from input data source.
  functions - aggregate, conversion, date + time -
  parameters - parameters passed.
  Cached lookup - lookup on data.

Azure Data Factory -
└ Author - Data flows - New Data Flow (Properties - Name
Publish
└ Add Source
└ Name
Select - Dataset
└ Data flow debug
└ spin up Spark cluster
└ Debug time to live
Debug Setting
└ Row limit
Data Preview
└ refresh

Destination
+ Sink
└ Name
Dataset in Synapse
Mapping
└ auto mapping - uncheck to confirm
Data Preview

Azure Synapse Analytics
└ Develop - New SQL script
└ CREATE TABLE table

ADF
└ Author - Datasets - New Dataset
└ Synapse
└ name
linked service
table name

- Data flow - New
└ Add Source

> + filter - name
filter on - Open expression builder
Data Preview - refresh
Save + Finish

> + Derived - name
Column column name, expression builder
Save + Finish

> + Sink - name
dataset - synapse table

Publish data flow.

' Author - Pipelines - new
' Drag Data flow
' Name
' Setting - Data flow name
- Staging linked service
Staging storage folder

- Validate
- Publish
- Add trigger
' trigger now or schedule
time

' Monitor - pipeline runs
' refresh

Synapse → Confirm data in table

~~APP~~
~~' Author - Pipelines - new~~

Monitor A pipeline
- set alerts
- Azure Monitor

ADF – pipeline – Delay
  └ use data flow runtime
    └ use delay duration

Details – Stages – lineage

Azure Synapse Analytics
  └ manage – linked services
  └ data – Linked
    └ synapse storage account
      └ upload – csv
        └ right click – Load to dataframe
  └ manage – Spark pool – new
    └ name
    └ node size, number

  └ develop – ~~notebook~~ ~~SQL script~~ – attach to ~~SQL~~ spark pool
    – read csv
    – create database table
    – save as table

  └ integrate – new pipeline – drag notebook
    └ name
    setting – select notebook
      spark pool

    └ validate
      publish
      trigger

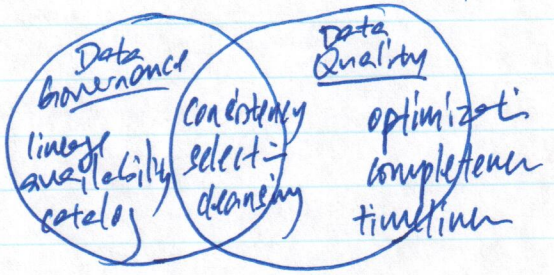
  └ monitor
    └ refresh

Power Query: (code-free, excel like transformations)
- uses M language
- ADF uses Apache Spark behind scenes, automatically converts M language types to Spark.

ADF
- Author - Power Query - New
  - Select Dataset
  - (Action similar to Excel)
  - ....
- Publish

- Pipeline - new - Power Query (drag)
  - name
  - Settings - select, staging
  - Sink - select - db.
- validate
- publish
- trigger

13. Azure Pipeline Data Quality



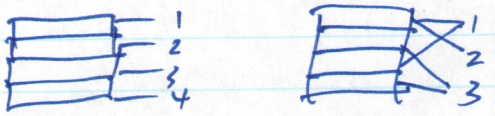
Data Governance
- track data lineage
- know who it's available to
- understand catalog of data available

Data Quality
- optimizing data so its complete, timely, consistent.

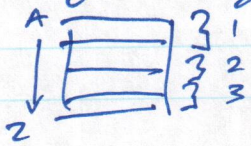
Best practices to ensure ADF & Synapse achieve limitless scale

1. Tuning of Integration Runtime
   - Spin up cluster for Spark compute takes 3-4 minutes
     if concurrent jobs, make sure 'Time to Live' is long enough, subsequent
2. Partitioning data
   1. Current partition - recommended for all except SQL db.
   2. Single partition - smaller dataset
   3. Set partition source - recommended for SQL db.

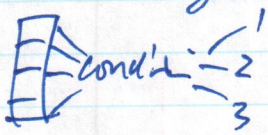
* Round Robin * Hash



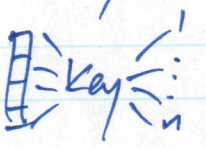
Dynamic Range



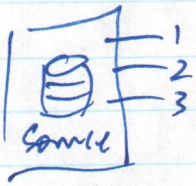
Fixed Range



Key



Source



3. Schema Drift
   - Columns in source data changes after Pipeline is developed + in use.
   - enable "Allow schema drift" in Data flows for source + sink dataset.

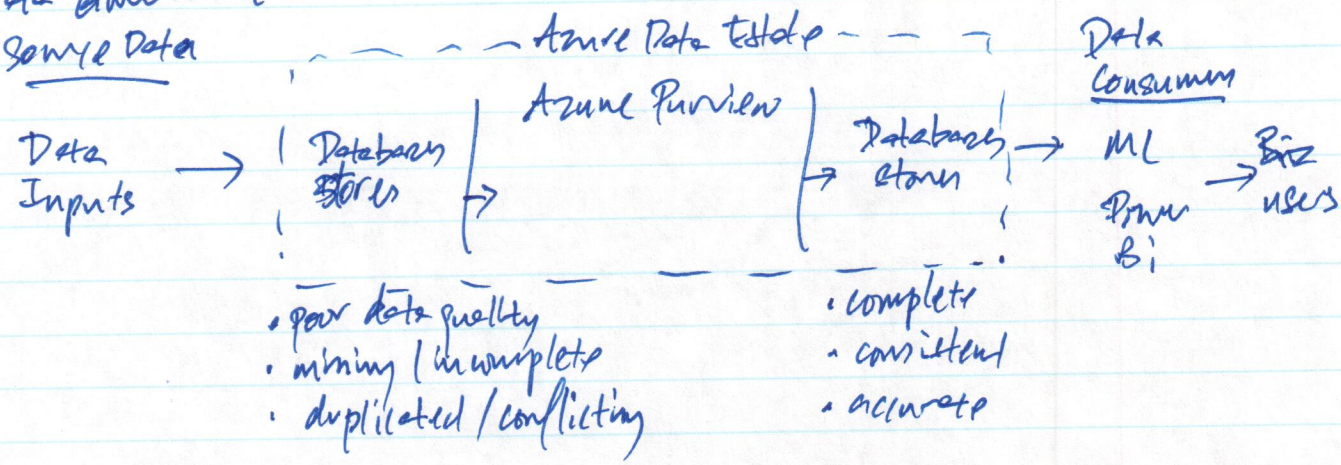
Slowly changing dimensions
- e.g. customer phone, address, product name.
- 6 types of SCD
  - 0 : ignore changes, keep only original
  - 1 : overwrite existing with new
  - 2 : add row with new values, add version column
  - 3 : add new columns for new values
  - 4 : maintain all historical values in new history table.

I: extension of Type, where a mini-dimension table is created by maintaining keys.

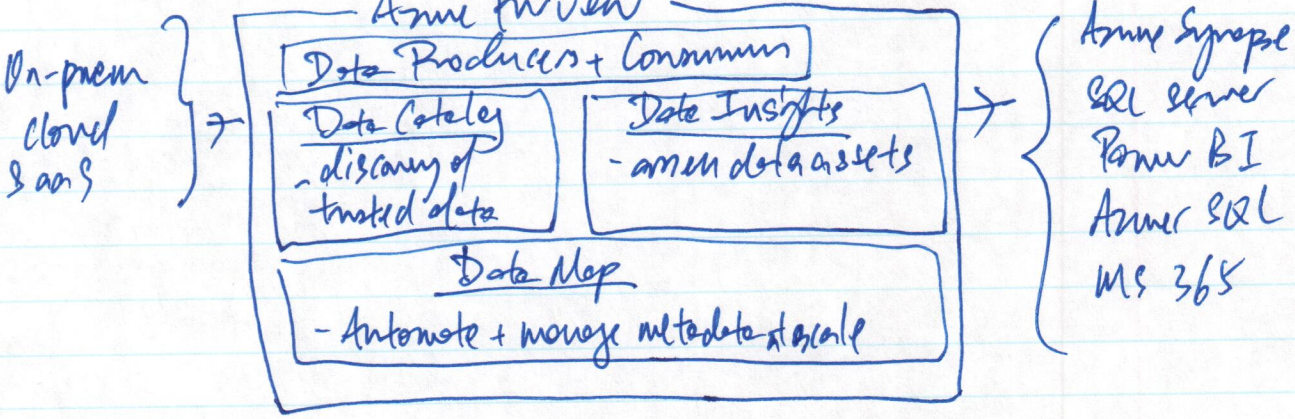
6: combination of 1, 2, 3 - adds new record for new values + maintains a new column.

4:

Data Governance:



Azure Purview -
- data governance service
- discover data quickly, classify data, data lineage.
- creates data map of data estate with datasets + their relationships + stores the corresponding metadata.
- connects with ADF, Synapse.



Create Purview account
↓
ADF
└ manage - Azure Purview - Connect to Purview account

└ author - pipeline - trigger now

Azure Purview - Open MS Purview Studio
└ Browse assets
└ collection
└ SQL
└ Lineage

14. Azure Data Pipelines in Production:

Parameterizing Pipelines:
- supported by pipelines, datasets, linked services, data flows.
- examples:
  1. linked service - database name, username
  2. pipeline - parameters passed from pipelines to data flows.
  3. global - used by all pipelines, a dataflow
     - Dev to QA to Prod env. pass to
     - only for ADF, not Synapse.
  4. System variables - @pipeline().DataFactory()
     @pipeline().RunId
     @trigger().startTime, ...

ADF
└ Data Flow - click on canvas
└ Parameters - New
└ name, type, value

use Parameter in Open Expression Builder

└ Pipeline - same as Dataflow to create
- to pass to Dataflow, ~~click~~ select Dataflow_Parameter
└ click on value - select Pipeline expression

Select Parameters

~~Global~~ Manage → Global Parameters - New
(available in ADF or Synapse)

- Linked Service - Select - Parameters - New

ADF, Synapse } JSON Scripts — Production
QA
Dev environments

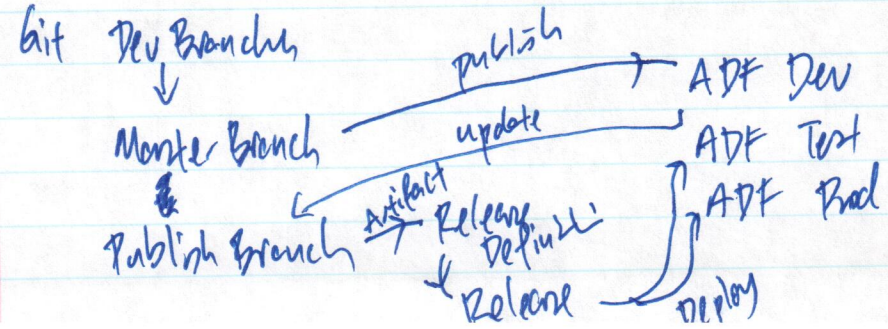
↓

Create with - CLI
Powershell
.NET
Python
REST
ARM templates

az datafactory create --resource-group ResourceGroupName
--factory-name FactoryName

Linked services - { } - copy JSON
Data flow - { } " "

CI/CD:
- integrate ADF, Synapse with Azure DevOps or GitHub
- use ARM templates for target environments



Create project in Azure DevOps Repo.

ADF
└ manage - Git configuration - configure
└ Azure DevOps
GitHub
└ Azure AD
└ Repo select
project name.
branch name

Author - all Pipelines
Datasets
Data flows
Linked services
Triggers
Integration Runtimes
} publish to repo

Project NYC payroll: