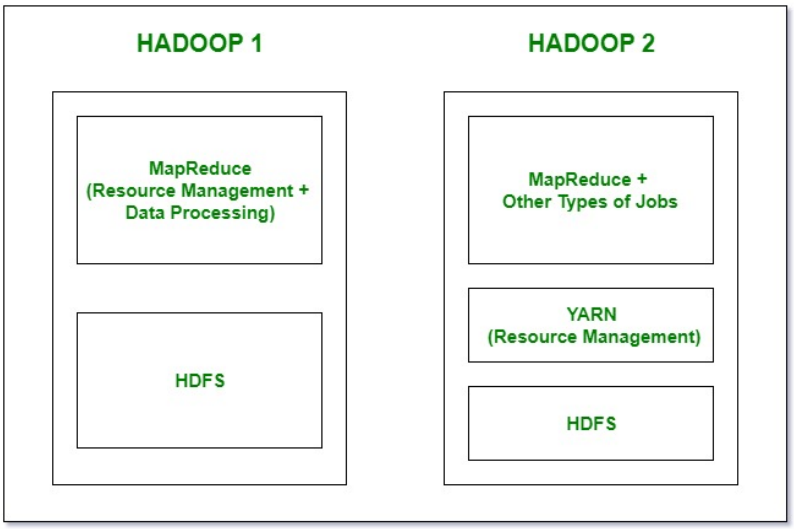
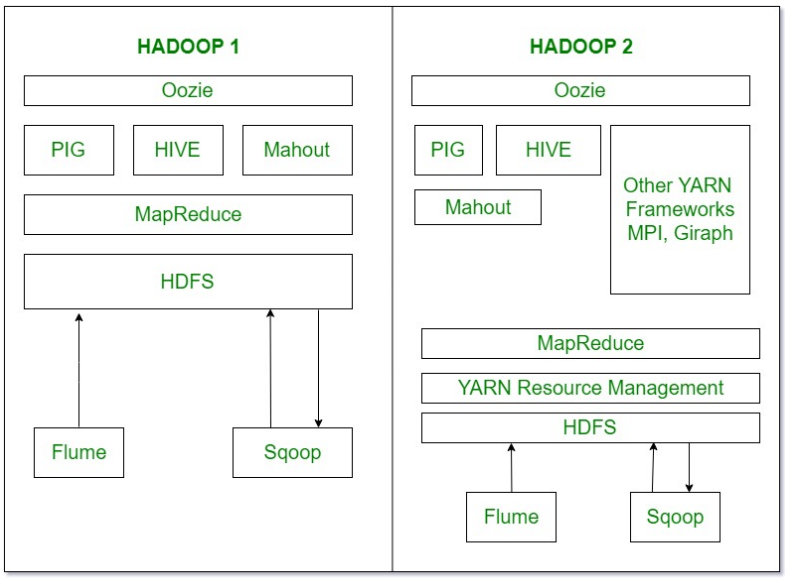
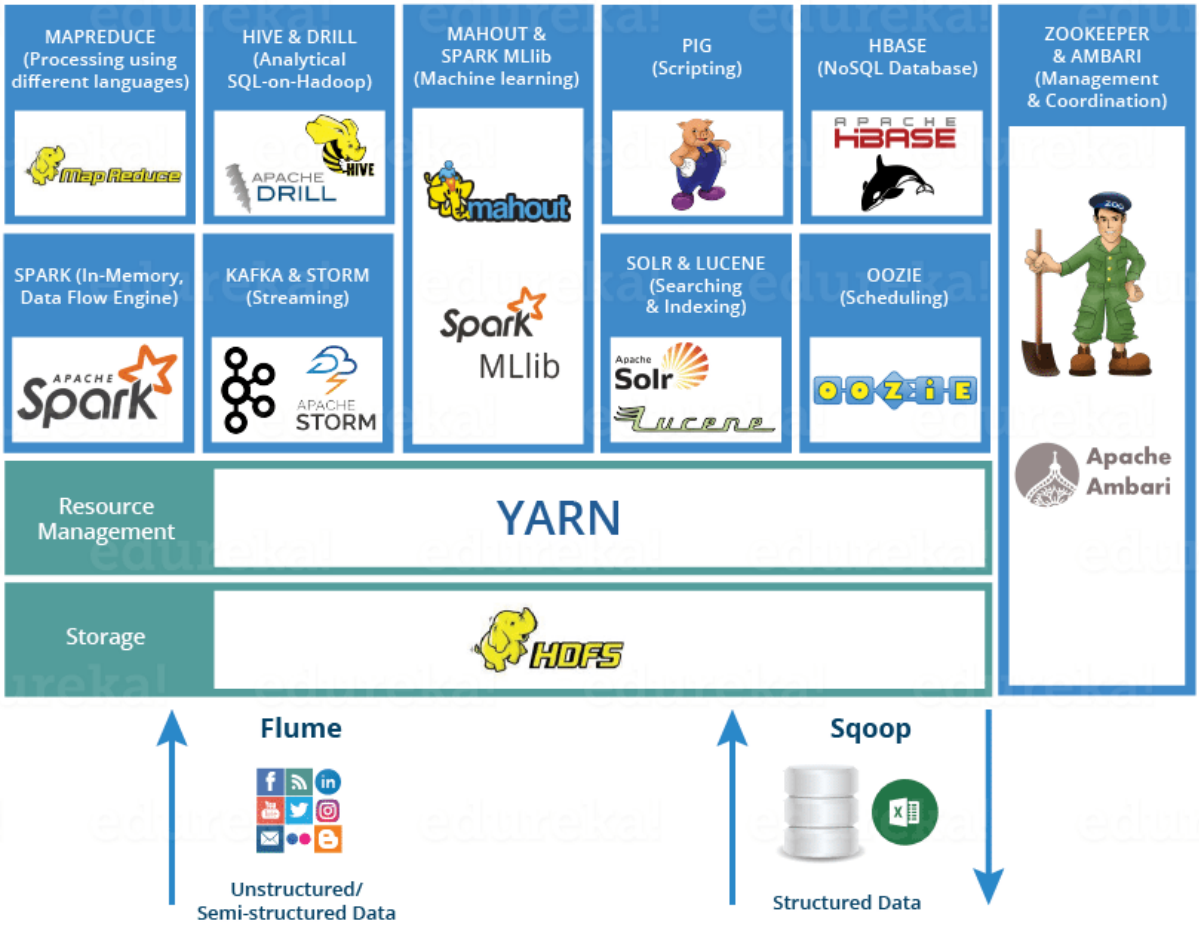
Ashpak Sheikh

Hadoop

HDFS + MAP Reduce
Hadoop engine name is Map Reduce
Spark engine name id Spark core (driver)
Spark Core is a multi language API (py , java, scala ,sql)
Zookeeper will manage all the hadoop components
When the mapreduce program is generated it get stored in local file system as .java







Below are the Hadoop components, that together form a Hadoop ecosystem

- HDFS -> *Hadoop Distributed File System*
- YARN -> *Yet Another Resource Negotiator*
- MapReduce -> *Data processing using programming*
- Spark -> In-memory Data Processing
- PIG, HIVE-> *Data Processing Services using Query (SQL-like)*
- HBase -> *NoSQL Database*
- Mahout, Spark MLlib -> *Machine Learning*
- Apache Drill -> *SQL on Hadoop*
- Zookeeper -> *Managing Cluster*
- Oozie -> *Job Scheduling*
- Flume, Sqoop -> *Data Ingesting Services*
- Solr & Lucene -> *Searching & Indexing*
- Ambari -> *Provision, Monitor and Maintain cluster*
- *Hbase will require zoo keeper as it is not involving mapreduce*

Hadoop modes :
1)stand alone
2) sudo
3) distributed (multi node)

RDBMS :
1) Most of them are running OLTP
2) Limited storage
3) Works on single server that's why it is slow

What is Big data?

Big data is a collection of data from many different sources and is often described by five characteristics: volume, value, variety, velocity, and veracity.

- **Volume**: the size and amounts of big data that companies manage and analyze
- **Value**: the most important “V” from the perspective of the business, the value of big data usually comes from insight discovery and pattern recognition that lead to more effective operations, stronger customer relationships and other clear and quantifiable business benefits
- **Variety**: the diversity and range of different data types, including unstructured data, semi-structured data and raw data
- **Velocity**: the speed at which companies receive, store and manage data – e.g., the specific number of social media posts or search queries received within a day, hour or other unit of time
- **Veracity**: the “truth” or accuracy of data and information assets, which often determines executive-level confidence

The additional characteristic of variability can also be considered:

- **Variability**: the changing nature of the data companies seek to capture, manage and analyze – e.g., in sentiment or text analytics, changes in the meaning of key words or phrases

Spark : Process data in memory (RAM)

Hadoop : Process data in Disk

Data Sources :

Static - RDBMS

Dynamic : continuous data, (flipkart server, online sites server, sensors etc)

MangoDB : NO SQL(mostly Json data) , can store any type of data

Big data approach -

1) Needs file system (HDFS)

What is the size of your HDFS?

- Ans we used ‘N’ node cluster #4 node
- Each node size - 1TB/etc (storage)
- What was the ram ? #32GB/node

- Name node(master Node) is also called as Meta Node
- Secondary Name Node also called as “Standby name node”
- In Name_node metadata gets stored in memory
- Block data info is stored fs_image , edit logs in Name node inside memory - **Meta files**

We learn about :

- What is the Need of Big data technology
- Difference between RDBMS and HDFS
- Difference between Hadoop and Spark
- How HDFS works
- Hadoop Master slave architecture
- Data sources
- Pay-as-you go provides Auto scalability (so when required number of nodes will be added automatically)
- In Hadoop 2.X we have High Availability there(HA) , Resource management(multiple jobs will run parallelly based on available resourcesresources)
- 2.X have YARN(yet another resource negotiator) will manage resources

- Config.num_nodes - to set number of nodes
- 1.X was running jobs one by one in queue

- Once the file system is ready and client grands the source storage permissions then we create a ETL pipeline to extract data from source to HDFS
- Once data into HDFS we start the analysis by processing data (parallel processing)
- The output from processing will be stored in HDFS
- Before sending the output file to client server SME will review the data
- Then BI tools will work on visualization

20-06-2023

- Block size is configured in name node
- Most highly used config files are
    1. Confi-site.xml
    2. HDFS-site.xml
    3. MapRed-site.xml
- HDFS files are immutable(offline file system)
- Most of the firms are using Cloudera(CDP) and MapR , Hardenburg (HDP) - These are Called Distributors
- With Hadoop we can do two types of analysis
    - Batch processing (Pig , Hive) - Complete data will be scanned(data stored as rows)
    - Online processing (HBASE(no sql DL , OLAP) , Hbase doesn't use map reduce it directly works on top of HDFS) column store - Schema less (in online processing only particular column will be scanned)
    - Streaming is not possible with Hadoop
- With using **Scoop** we can bring data from any static source to HDFS On top of MR - default ETL tool(data ingestion only)
- To get data from any live server(dynamic data) to HDFS we can use **Flume** on top of MP - Default ETL tool (data ingestion only)
- Informatica , talent , data stage - **3rd party ETL tools**
- One Job tracker for a cluster - v1
- Each DataNode has one Task tracker - v1
- Task Tracker runs the process on the blocks -v1
- Number of blocks = number of map task
- In v2 Job Tracker- Resource Manager , Task Tracker -> Node Manager

Types Storage:

★ Data Lake :
Its stores data of any format data (big data file system)
★ Data warehouse :
Stores only Structured data
★ Data Base:
Structures data of limited size

While data warehouses store structured data, a lake is a centralized repository that allows you to store any data at any scale

Examples of data warehouses include:

- Amazon Redshift.
- Google BigQuery.
- IBM Db2 Warehouse.
- Microsoft Azure Synapse.
- Oracle Autonomous Data Warehouse.
- Snowflake.
- Teradata Vantage.

A data lake is a repository of data from disparate sources that is stored in its original, raw format. Like data warehouses, data lakes store large amounts of current and historical data. What sets data lakes apart is their ability to store data in a variety of formats including JSON, BSON, CSV, TSV, Avro, ORC, and Parquet.

You might be wondering, "Is a data lake a database?" A data lake is a repository for data stored in a variety of ways including databases. With modern tools and technologies, a data lake can also form the storage layer of a database. Tools like Starburst, **Presto**, Dremio, and **Atlas** Data Lake can give a database-like view into the data stored in your data lake. In many cases, these tools can power the same analytical workloads as a data warehouse.

The following are examples of technology that provide flexible and scalable storage for building data lakes:

- AWS S3
- Azure Data Lake Storage Gen2
- Google Cloud Storage

Other technologies enable organizing and querying data in data lakes, including:

- MongoDB Atlas Data Lake.
- AWS Athena.
- Presto.
- Starburst.
- Databricks SQL Analytics.

HDFS:

Slogan - Write once and read multiple time

- HDFS is a file storage system only
- Default file size of HDFS is text file (Flat file) , not changeable - By vainket sir
- Every block is a 1 task (no of block = no of task)
- Single Data node can execute multiple task
- Complete execution of the tasks(program) is job

What file format is used in HDFS?

HDFS file formats supported are Json, Avro and Parquet. The format is specified by setting the storage format value which can be found on the storage tab of the Data Store. For all files of HDFS, the storage type (Json, Avro, Parquet) are defined in the data store.
- By Apache Software
CLI - command line interface

- HDFS URL **- ip://50070 -** to see HDFS UI
- http://localhost:50030/jobtracker.jsp - Admin url to see running jobs/resource managers\
- HUE (hive user interface)
- Write 'Hive' in command prompt to enter into Hive
- Write 'HBase shell' to enter into HBase
- Write Pigl' to enter into Pig , it will come as 'grunt'
- Zookeeper-client
- Beeline to get into beeline (for hive cli)
- Ctrl+c -> use this to get into home directory from any tool in CLI
- hadoop fs -mkdir milesdata - create a directory in hdfs
- hadoop fs -mkdir milesdata/futurensedata
- **hadoop fs -rmr milesdata/futurensedata - remove directory**
- **hadoop fs -put /home/training/brave.txt /user/training/milesforce - to copy file**
- **hadoop fs -ls /user/training/milesforce**
- **cat>brave.txt**
  how are you bro? Ctrl+c to exit
- **hadoop fs -cat /user/training/milesforce - to read file**
- **hadoop fs -rmr /user/training/milesforce - to remove file**
- **hadoop fs -put sale* trial1 to move all the files starting with name 'sale'**
-

Hadoop 2.X

- http://localhost:8088/cluster - cluster manager/Yarn
- http://localhost:50070/dfshealth.html#tab-overview HDFS UI - utility->Brows the file system
- http://localhost:8042/node - NodeManager
- **localhost:8888/about/#step2 - > hive web interface**
- **Beeline**
- **hadoop dfsadmin -report**

- **hadoop fsck / -blocks - will show data about block/corrupted blocks**
- **hadoop dfsadmin -safemode get -> to check safe mode status**
- **Safe mode will stop reading and writing permissions**
- **hadoop dfsadmin -safemode enter -> to enter in safe mode**
- **hadoop dfsadmin -safemode leave -> to exit safe mode**
-
-

- In hadoop 1.X block size was 64MB
- In Hadoop 2.X block size is 128MB

Advantage of Block Size :

- Fixed size - easy to calculate how many fit on a disk
- A file can be larger than any single disk in the network
- If a file size is less than the block size , then the only needed space will be used and remained space in block will be used to store other data

HDFS is built using **the Java language**; any machine that supports Java can run the NameNode or the DataNode software.

Map Reduce

- Process data
- Map reduce program is use to analyze data
- Whenever reading anything from the file, all the blocks of these file will execute
- With using **Scoop** we can bring data from any static source to HDFS On top of MR - default ETL tool(data ingestion only)
- To get data from any live server(dynamic data) to HDFS we can use **Flume** on top of MR - Default ETL tool(data ingestion only)
- Informatica , talent , data stage - 3rd party ETL tools
- Oozi is use to schedule jobs - default from hadoop
- Airflow is also there to schedule jobs

When getting data with ETL from the client source by default 4 Map task will run
And 0 reducer , as we are not doing transformation .
When we are getting bulk transfer from client to HDFS by default Scoop will use 4 Map tasks by default / File(table) , but it is better to use 1 map task to reduce load on Name Node.

When data is very big(ex 10GB) then only go with 4 map tasks.
If task map task is not completed within 3 sec then task tracker will start another task for the same data in other node, so whichever task succeeds first that data will be taken and other slow task will be killed , by default there will be 4 duplicate task during slow task , if all 4 task runs slow then job will be failed . this scenario is called **Speculating Execution**

Hadoop 2 :

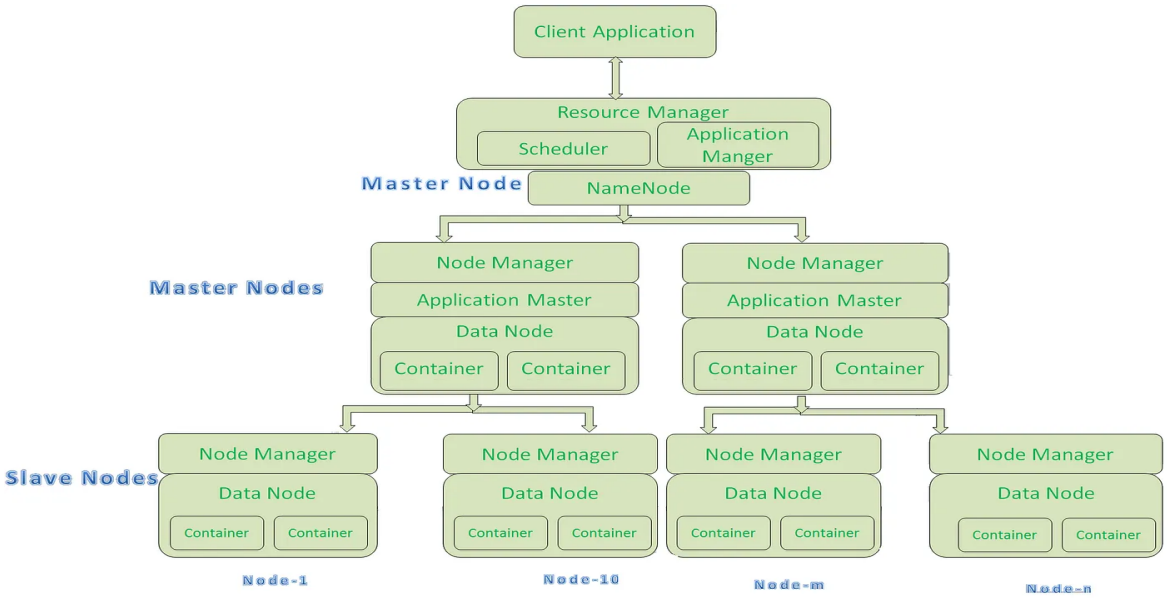
- ➔ Yarn = MapRed + Node Manager
- ➔ 1 Application Mater per job

Resource manager(resource manager 2) is connected with (Active Name node and Standby Name Node)

- Resource configuration happens in Resource Manager (RAM, Processor , Storage)
- Resource manager allocated the resources to the node managers

Let's say we are running one Hive query(job) :

- Request will go to RM it will release AM(Application Master)
- AM will request Resources(RAM +CPU) to RM
- Now AP manager allocates the Resources to the Node managers
- Now node manager will execute the task(map tasks) on Data Nodes
- After all map tasks are successed reducer job will run
- Once the job is succeed and acknowledged to RM the resources allocated will be released
- Application master will be running on any Node Manager



Hadoop 2.x In-Detail Architecture

Data Ingestion Tools (ETL) :

1. Scoop
2. Flume
3. Kafka
4. Informatica
5. Data Stage

Scoop :

Two modes :

1) Append
2) Last modified

Data Ingestion

- Use to import and export data
- Default Map tasks are 4
- Default reducers are 0
- Use to transfer data from any RDBMS to HDFS

Analysis

- When we do data analysis in Hadoop cluster *no. of blocks = no. of MAP tasks*
- But default reducer is 1

Hint : Hive uses default Storage as HDFS and stores data as tables so better to ingest data into hive database so we can run queries on top of it.

Open terminal in cloudera
Transfer single table from server to HDFS
sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table customers -m 1

Transfer multiple table - Transferring all files to imported_tables directory
sqoop import-all-tables --connect jdbc:mysql://localhost:3306/training --username root --warehouse-dir imported_tables -m 1

Single table to given location
sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table ordersop --warehouse-dir imported_tables --split-by order_id

–split-by column name :-when we don't have primary key

HDFS to client server

- first create a table in client database then export data

sqoop export --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table result1 --export-dir imported_tables/ordersop/part* --input-fields-terminated-by ','

sqoop eval --connect jdbc:mysql://localhost:3306/orders_db --username root --password cloudera --query "select * from result1 limit 3";

sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --table result1 --m 1 --where "order_id > 1000" --target-dir /retail_db

List client Data bases:
sqoop list-databases --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera

List tables
sqoop list-tables --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera

scoop incremental to **append** new records in HDFS:
sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table customers -m 1 --**incremental append** --check-column customer_id --last-value 12435
It will add new part file into customers in HDFS

Create Incremental Last Modified Job link
Steps to merge parts

- Create .jar file with updated data

- Get incremented data into HDFS
- Use merge command on both the parts to merger

sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table orders --target-dir orders1 --**incremental lastmodified** --check-column order_date --last-value '2014-07-24 00:00:01' -m 1 --append

Jar file creation
sqoop codegen --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table orders --outdir codegen_orders1/

Merging
sqoop merge --new-data orders1/part-m-00001 --onto orders1/part-m-00000 --target-dir ordersmerge --jar-file /tmp/sqoop-cloudera/compile/e3f89965b4b2c6bcc1aaacaa350695cf/orders.jar --class-name orders --merge-key order_id

2nd Practial:

sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table job_increment --warehouse-dir increment_data -m 1

- now add new data to database table

sqoop codegen --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table job_increment --outdir increment_data/job_increment

/tmp/sqoop-cloudera/compile/1ff3e50a2f90651383d5b9c3c08b2e5e/job_increment.jar

- Now add incremented data part into HDFS

sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table job_increment --target-dir increment_data/job_increment --incremental lastmodified --check-column update_time --last-value '2022-01-03 00:00:01' -m 1 --append

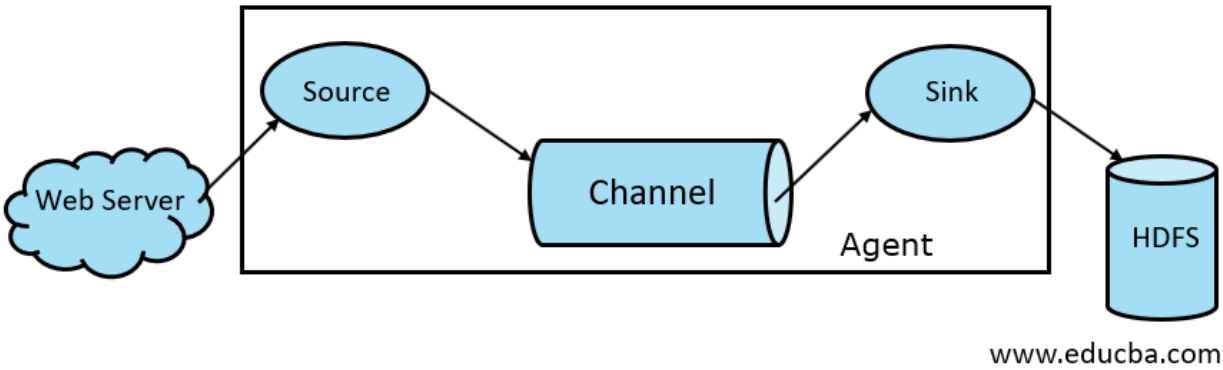
sqoop merge --new-data /user/cloudera/increment_data/job_increment/part-m-00002 --onto /user/cloudera/increment_data/job_increment/part-m-00000 --target-dir job_increment_merge --jar-file /tmp/sqoop-cloudera/compile/1ff3e50a2f90651383d5b9c3c08b2e5e/job_increment.jar --class-name job_increment --merge-key id

When using more than 1 reducer we use partitioner which decides which keypoint to which reducer (runs before reducer)

- Pickling in pyspark - Use to increase the performance of analysis
- Memory management in python
- Name space in python -A namespace is a system that has a unique name for each and every object in Python. An object might be a variable or a method. Python itself maintains a namespace in the form of a Python dictionary.

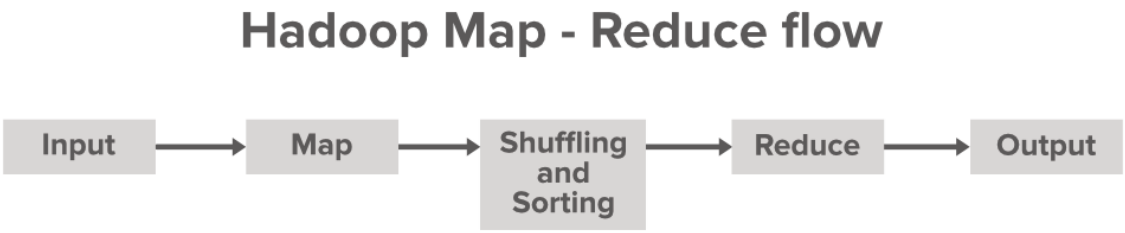
- Decorator in python
- SQL - **Materialized vs normalized** view - Materialized view stores the result in the physical memory,
  Normalization views- are read-only once created, and cannot be edited. Data normalization is the process of converting data to a common form so that it can be quickly retrieved for viewing and reporting purposes such as production accounting.
- How to debug code in python
- **Is** vs **==** in python -> == compares two values whereas 'is' checks memory location
- What is metaclasss in python : **Python** supports a form of metaprogramming for classes called **metaclasses**. **Metaclasses** are an esoteric OOP concept, lurking behind virtually all **Python** code.
- How will you share variables across the models in python?
- // integer division
- How to perform static analysis in python
  Static analysis is a method of debugging that is done by automatically examining the source code without having to execute the program.
- What are the different NoSQL db availables? - MangoDb, redshift , cassandra
- Database drivers for sql , mysql, postgre
- * schema vs Snowflake Schema
- * schema - All the dimension tables will be connected to fact table
- OLTP -star schema
- Different Data models - * schema and Snowflake Schema
-

Flume:



Map Reduce :

While running ETL pipeline and merging data then reducer will be 1.



There is inbuilt **Combiner** to improve performance (it combines the key value of map output)
Works when there is single reducer , else there is **partitioner** which partition data for reducers

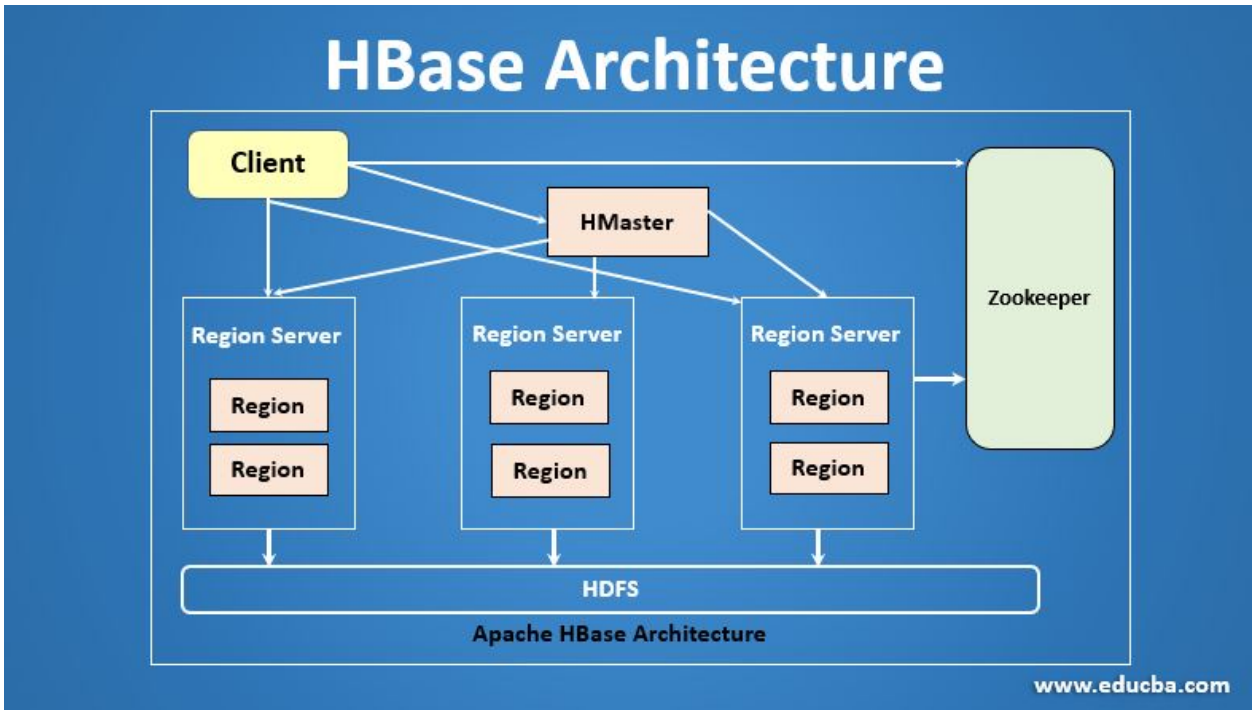
Case study : write the map reduce program to find the total bill of each patient

1) Copy all the data from training case study to VM
2) Open Eclipse , create new project and create 3 files (conf_bill , maper_bill , reducer_bill)
3) Copy data from downloaded data(codes) in above files
4) Run conf_bill as java application
5) Now export jar file by right clicking on project name and select jar file and then export

6) Now run this command in terminal " **hadoop jar /home/training/ibm1.jar /user/training/host.txt /user/training/out_redu/ "**
7) Hadoop jar jar_location text_file_loc output_location(hadoop)

Case 2 : month wise ip address:

1. Move file to hdfs - hadoop fs -put /home/training/ab.txt new_data
2.

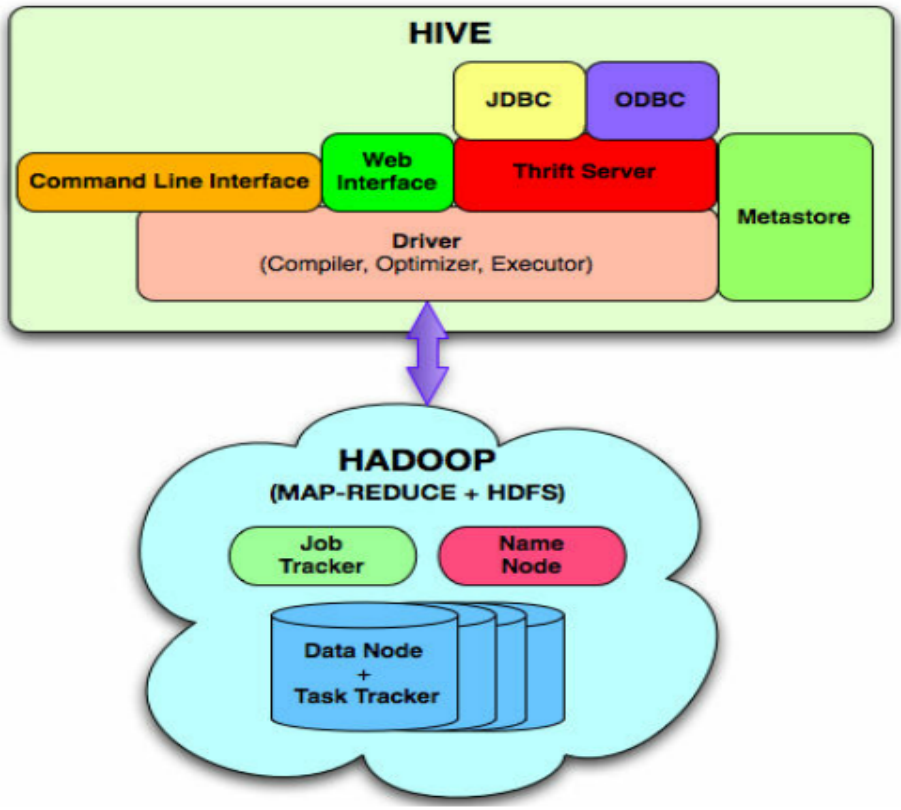


Hive:

Hive is a data warehouse(data stored as a Table-Structured) which is inbuilt on top of HDFS

- Hive is used in batch processing analysis
- Hive default storage is HDFS
- So when we run hive query mapreduce program will be executing
- Hive is not OLTP but hive is like SQL called HQL
- Hive is big data warehouse (any large volume of Data)
- Hive is open source
- Hive is Offline process
- No need to mention size of the data type while defining schema

- Need to install Mysql or any SQL on cluster to store metadata info in RDBMS(*Derby* default id no RDBMS is installed)
- By default Hive does not Support ACID property
- Hive works with structured and semistructured data
- **Types of Indexing in hive** - > Compact indexing and Bitmap Indexing
- H- Catalog - to join hive with HBase



Thrift server:

- To check query syntax

MetaStore:

- Stores metadata
- It is connected with RDBMS to store Schema (Internal **Derby** if RDBMS is not installed)
- That means Hive stores Metadata at RDBMS

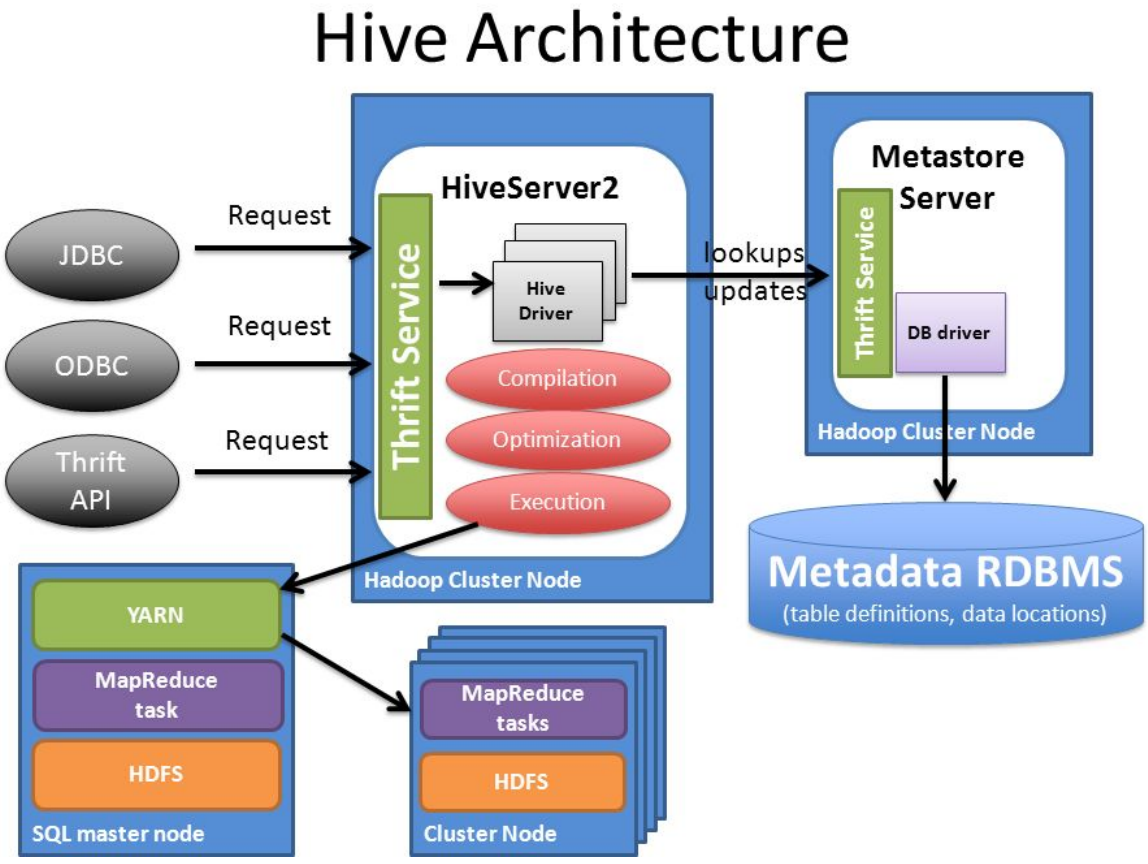
Fastest Query Language:

- Delta Lake
- Spark SQL

Hive data Type:
There are two different data types in hive

- Primitive
- Complex data type

Primitive data type :
Int, float , double , string
Complex data types :
Struct , Map , Array



Hive Commands:

Hive will support all SQL commands:
DDL, DML
Except - insert , update , delete - till HIve 1.2x

After 1.2x -
It supports insert, update, delete - but not by default
We need to activate acid libraries to use ACID properties

Remember the versions of spark , python , hive etc

Hive data warehouse:

1. Create pipeline from **source to the target Hive** data warehouse

Sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username root --password cloudera --table customers --hive-import -m 1
Note - *No need to create table in hive , table will be created automatically with the same table name*

Import all tables to Mysql

Sqoop import-all-tables --connect jdbc:mysql://localhost:3306/mysql --username root --password cloudera --hive-import -m 1

To copy flat file from HDFS to hive we need to create Table first in hive with delimited

create table mybills(doj string , pname string , roomnum int ,dbill int ,mbill int ,nbill int , obill int) row format delimited fields terminated by ' ';

Load data HDFS to Hive

load data inpath 'milesdata/host.txt' into table mybills;

Once the data is moved from milesdata/host.txt the file from HDFS will be gone

#above table is called as internal tables (managed tables)
There are two types of tables in Hive warehouse:

1. Internal tables (managed table)
2. External table (unmanaged tables)
   We create external table to store outputs so if by accident the table gets drop then only metadata will get deleted and data will be there in the file

1)Internal tables (managed table)

The tables which are stored in warehouse are called internal tables , if you drop internal table then metadata will also be dropped

Use describe to see table schema

External table:

Create external table : hive

create external table myop(pname string , total_bill int) row format delimited fields terminated by ' ' location '/user/hive/warehouse/myop/results.txt';

Insert data into external table:

insert overwrite table myop select pname , sum(dbill + mbill + nbill +obill) from mybills group by pname;

Hive to RDBMS :
Insert into / insert overwrite

Drop table :
Drop table myop;

Now we can see the file is still there in HDFS and only metadata got deleted.
hadoop fs -cat /user/hive/warehouse/myop/results.txt/00*

We can see data is still there

Now send the file to client (in this case it is my local directory)
hadoop fs -get /user/hive/warehouse/myop/results.txt/00*

What we did yesterday :

- Saw hive architecture (components)
- Data type in Hive (Primitive , Complex)
- Created hive internal and external table by getting data from Source(mysql)
- Difference between internal and external table
- Where the the meta data gets stored for hive table
- Hive warehouse uses HDFS in the backend
- What happened when we drop internal and external table

28-06-2023

Agile methodology :

- Stand up called
- Review meetings
- Client can change requirement at any time

Scrum board -> Do - Doing - Done
Product Owner -> Scrum Master -> Scrum Co-ordinator -> BA, DE, Tester , Devops

Hive analysis Steps:

1. SQL or any SQL >>>> Hive DW or HDFS
2. Create table and load files from hdfs to hive
3. Run hive queries as per the client problem statements
4. Verify output data
5. Create external table and overwrite output data to the external table
6. Send the output table to the client server/storage
7. Tableau/PowerBI with SQL >>> report

Export data to client server from Hive:

```
sqoop export --connect jdbc:mysql://localhost:3306/retail_db --username root --password
cloudera --table xyz --export-dir /user/hive/warehouse/myop/results.txt/0*
--input-fields-terminated-by ' '
```

Case study 2:

Load data into HDFS From local:

```
hadoop fs -put House_Survey.csv
```

Now create table in hive and move data from HDFS to HIVE:

```
Create table house_survey(ID int,Size_of_House string,Number_Of_Bathrooms
int,Age_Of_Building string ,Number_Of_Balcony int,Number_Of_Fireplace int,Furnished_Status
string,Car_Parking string,Air_Conditioning string,Private_Garden string ,Lift_Available
string,Area_Status string ,Rent int) row format delimited fields terminated by ',';
```

Now load data into hive table:

```
load data inpath '/user/cloudera/House_Survey.csv' into table house_survey;
```

```
Select Area_Status ,Number_Of_Balcony , sum(Rent) as total_rent from house_survey group
by Area_Status ,Number_Of_Balcony order by total_rent  desc;
```

```
with cte as
(Select size_of_house ,Area_Status ,Number_Of_Balcony , max(Rent) , sum(Rent) over() as
total_rent from house_survey group by size_of_house,Area_Status ,Number_Of_Balcony
)
```

```
, high_amount as (
select size_of_house, Area_Status,Number_Of_Balcony,max(Rent) as max_rent from
house_survey group by size_of_house,area_status ,number_of_balcony)
, combined as (
select a.size_of_house ,a.Area_Status , a.Number_Of_Balcony , a.total_rent ,b.max_rent
from cte as a join high_amount as b on a.Area_Status = b.Area_Status and
a.Number_Of_Balcony = b.Number_Of_Balcony and a.size_of_house = b.size_of_house )
select * from combined
```

Create Big Bazar Hive table:

1) hadoop fs -put bigbazarsales.csv

2) set hive.exec.max.dynamic.partitions = 8000;
3) set hive.exec.max.dynamic.partitions.pernode=8000;
4) Create table bigbazar(Region string ,Country string,Item_Type string,Sales_Channel string,Order_Priority string,Order_Date string,Order_ID int,Ship_Date string,Units_Sold int,Unit_Price float,Unit_Cost float,Total_Revenue float,Total_Cost float,Total_Profit float) row format delimited fields terminated by ',';

5) load data inpath 'bigbazarsales.csv' into table bigbazar;

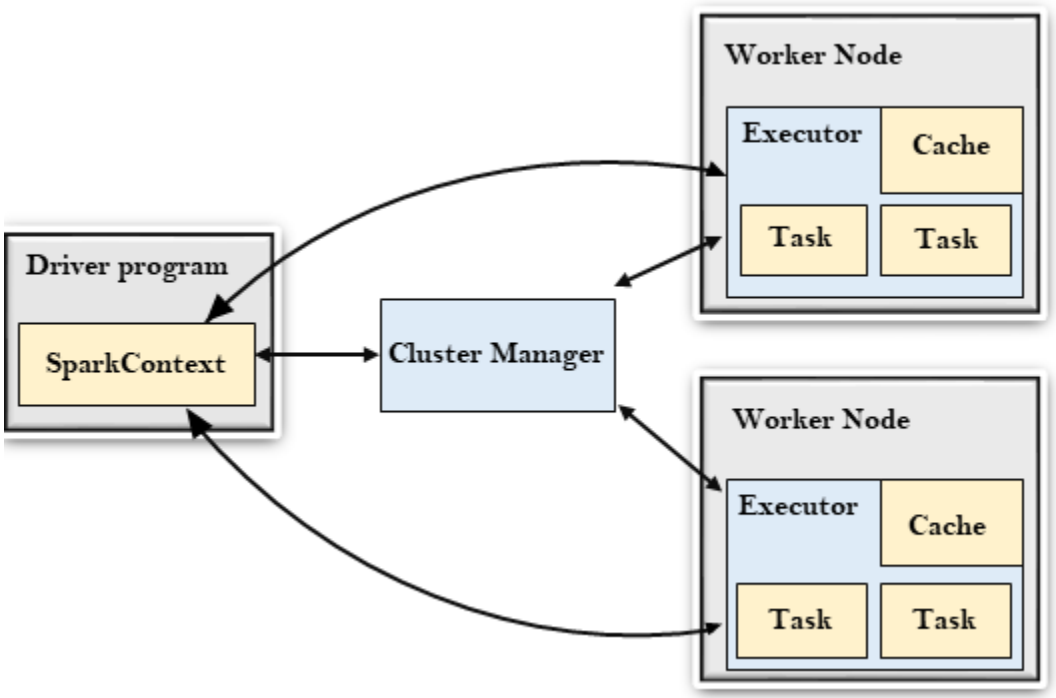
set hive.exec.max.dynamic.partitions = 8000;

Questions :
Hive version : 1.3x

- Hive doesn't support OLTP
- Max size of string supported in hive is 2GB
- When we run select * then why map reduce doesn't work in the backend ? - because we are not doing any aggregation function (also in the *where* condition)
- When do we use complex data types ? - when we want to represent semistructured data
- Jdbc - mysql , odbc - oracle server
- Hive vs Hbase - hive- batch processing , row stored, Hbase- online processing, column stored
- Hbase is an independant API that's why it is directly working on HDFS
- Types of partition tables - Static Partitioning and Dynamic Partitioning.
- Static partitions- it is created on HDFS file , dynamic partitioning - it creates partitions on table
- Max partition limit in hive - 500K in multi node cluster
- Can we drop partitions in hive - yes we can drop partitions (ALTER TABLE tablename DROP command.)
- Disadvantage of dynamic partitions - memory allocation
- Can we create partitions in an external table ? - yes , it works in the same manner as internal table
- Data pipelines- earlier oozie and now Airflow
- What is partitioning in map reduce and hive? - The partition phase takes place after the Map phase and before the Reduce phase. The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.
- Partition vs bucketing - partition is stored as a directory bucketing column stores as a files

Spark

Spark Architecture :



Always check number of partitions in RDD before starting the execution
Task is the number of partitions.
No of Stages - number of wide transformation + 1
Job is a complete execution of a program
Types of transformations : 2 - Wide transformation and Narrow transformation
Narrow transformation - No movement of data between the data between nodes, map , FlatMap
Wide transformation - Movement of data between nodes - reduceBykey , AggregateByKey
Important operations in spark : Transformations and action
Map vs flat map
Map - apply function on every element of data
Flat map - split the data , usually result into more number of rows (use case - array , map data type column)
What is the use of MapPartitions: No movement of data
ReduceByKey and reduce
Reduce - single value output
ReduceByKey - key wise aggregation , uses map side combine that's why it it more optimised
Default executors in spark- depend on the file size though default executors is 2
How to run a job in spark - we use Spark submit to run a spark job

Spark modes - *stand alone , cluster mode , client mode*
Cluster mode - driver inside resource manager
Client mode - driver is on the client side
All transformations happen Inside (shuffle ,transformation)- > Execution memory
Persist, Cache -> uses storage memory
~~Sort by - Global sorting , orderBY local sorting~~
order by- does whole ordering.
sort by: partition wise ordering.
Problem with spark :- Data Skew , shuffling , garbage collection , lazy evaluation , data spill out
Local[n] - for getting number of executors
SQL context - can also create Dataframe

- **Operator** package we use in RDD to import ROW function

Can not create RDD with parallelize function while reading data from HDFS
Data sets vs dataFrame - data sets have inbuilt encoders
3 important data abstraction to analyze data in spark - RDD , DataFrame , Dataset
Spark components - MLib , spark streaming , Pyspark
SparkSQL - Analysis
Streaming - Pyspark
Processing -
Data ingestion used in your project - Scoop , Floom , Kafka
reduceByKey vs AggregateByKey
Partitioning in spark - divide data based on file size
String function to divide data - split
Map input key values - key - line number and value is text
SortByKey vs Sort
Two important keywords when we want to run spark job
Map vs Map partitions
Map - apply function to each line
Map partitioning - it apply function only specific partition
Increase parallelism - we can add more executors to do it
Static allocation - when we specify number of executors(fixed)
Dynamic allocation - By default , spark allocated based on file size

1 HDFS block = 1 RDD partition - > True

Cluster mode - spark submit -- master ip:port number filename local[4]
Local mode - spark submit filename

Spark optimization techniques In Spark-

Partitions , cache , number of executors , shared variables , broadcast join(broadcasting small df to all nodes) ,serialization

DAG and how many Dags -> number of actions
No of Stages = number of wide transformations + 1?
What is lazy evaluation -> transformations won't be done until the action is called
Spark is extremely fast because of lazy evaluation , so when you are using persist it won't happen it will persist in the disc, so why do you want to persist in the first place??

- We are not using percist directly in the first place , when we are doing multiple transformations the results of those transformations will be stored in the temp RDD in the memory and then we can club it , and that's how we do the heavy lifting transformations

groupByKey() is just to group your dataset based on a key. It will result in data shuffling when RDD is not already partitioned.

reduceByKey() is something like grouping + aggregation. We can say reduceBykey() equivalent to dataset.group(…).reduce(…). It will shuffle less data unlike groupByKey().

aggregateByKey() is logically the same as reduceByKey() but it lets you return results in different types. In other words, it lets you have an input as type x and aggregate result as type y.For example (1,2),(1,4) as input and (1,"six") as output. It also takes zero-value that will be applied at the beginning of each key.

`persist()`:

- `persist()` is a method in Spark that allows you to explicitly specify the storage level for persisting an **RDD or DataFram**e.
- You can choose from various storage levels based on your requirements, such as `MEMORY_ONLY`, `MEMORY_AND_DISK`, `MEMORY_ONLY_SER`, `DISK_ONLY`, etc.
- The `persist()` method is more flexible compared to `cache()` since it lets you define the storage level that suits your needs.
- Persisted data is kept in memory or disk (depending on the storage level chosen) until explicitly unpersisted or until the Spark application terminates.

`cache()`:

- `cache()` is a shorthand for `persist(StorageLevel.MEMORY_ONLY)`, which means caching the RDD or DataFrame in memory only.
- By default, `cache()` stores the data in memory but can spill to disk if memory usage exceeds the available space.
- Caching using `cache()` is a convenient way to persist RDDs or DataFrames in memory, especially when you don't require explicit control over the storage level.
- The cached data is kept in memory until explicitly unpersisted or until the Spark application terminates.

Both `persist()` and `cache()` are lazy operations, meaning the caching or persistence happens when an action is called on the RDD or DataFrame. Until an action is triggered, the data is not cached or persisted.

There are two types of shared variables (read only variables)

1 - Broadcast variable

2 - Accumulators variable

Broadcast variable :

Immutable

Imagine you have a big dataset or a lookup table that you need to use in your Spark program. Normally, each worker node in the Spark cluster would need a copy of that data, and it would be sent over the network multiple times.

But with broadcast variables, you can send that data just once to all the worker nodes and have it stored in memory on each node. This way, when each worker needs to use that data, it can quickly access it from its local memory instead of repeatedly transferring it over the network.

Broadcast variables are useful for sharing large read-only data across tasks efficiently. They can be used for things like lookup tables, machine learning models, or any data that is big and doesn't change during the computation.

By using broadcast variables, you can reduce the time and network traffic required to share data across worker nodes, making your Spark program faster and more efficient.

```
broadcast_data = sc.broadcast(my_data)
```

`Broadcast_data.value` to access shared variable(return in the form of list)

These variables are using to implement the optimisations of RDD process or batch processing analysis

Accumulators variable:

“mutable”

Are shared variables that is used with RDD and Dataframes to perform aggregation like sum , count , avg

Variables are shared by all executors to update and add informations through aggregation

What is the worker Node in Spark ?:

Worker node is a slave node that processes the task submitted by spark Driver node

Cluster manager allocated the resources to each worker node in the form of executors , CPU codes and RAM

Executors are also called JVM’s

Rdd for batch processing and Data frame for interactive processing

RAM :

- **Heap memory** - portion of the memory controlled by the executor's JVM process . this is split into

  Storage memory , executor memory , user memory , reserved memory

- **Off heap memory** - portion of memory outside of the executors JVM's , which is free for garbage collection,(dataframes gets stored here)

Spark Tuning /Optimisations:

There is a one more optimization technique in pyspark

Inbuilt serialization is java serialization(takes high buffer ram)

Pyspark serializations

1. Pickling serialization (slower)
2. Kryo serialization(take low amount of buffer ram, faster)
3. Java Serialisation(default serialisation)

1 HDFS block in local = 1 RDD partition in local -> True

1 cluster = 1 machine -> True (single mode , pseudo mode)

We can use *local* keyword for number of processors we need

No of task = no of partitions in RDD - True

How many cluster master in spark cluster =1

Ho do we represent spark driver - spark context

Spark context is entry point to create RDD

Spark submit in client and spark submit master in cluster node

ORC compresse data 70%-80%

SparkSQL - if we get error we import 'SQL context' library

For Hive we use Hive Context

For streaming Streaming Context

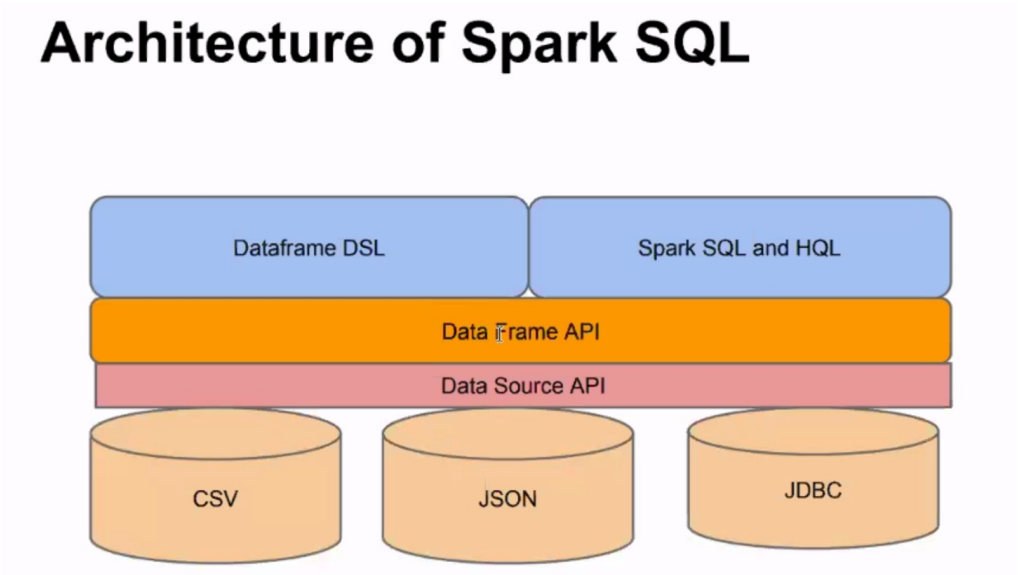
How to handle Multiple delimiters in PySpark?

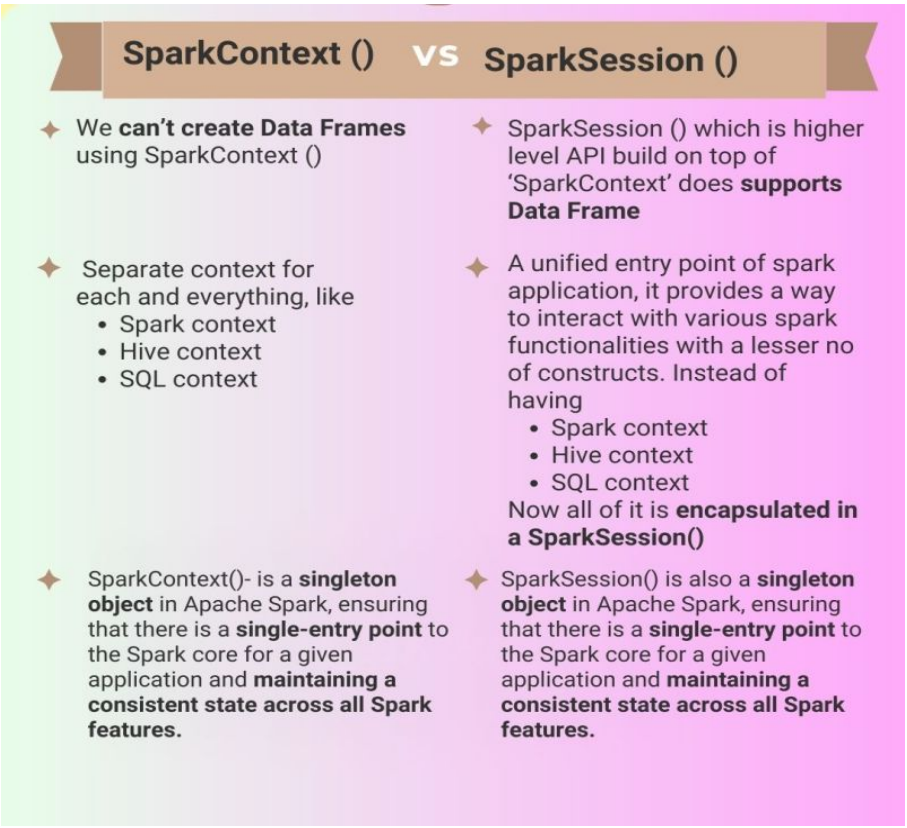
how to get number of rows for each partition in spark dataframe?

Pyspark.sql.functions import monotonically_increasing id for surrogate key (hiding information by replacing information by surrogate keys)

Why we should not use CRC32 for surrogate keys generation

Spark SQL architecture:

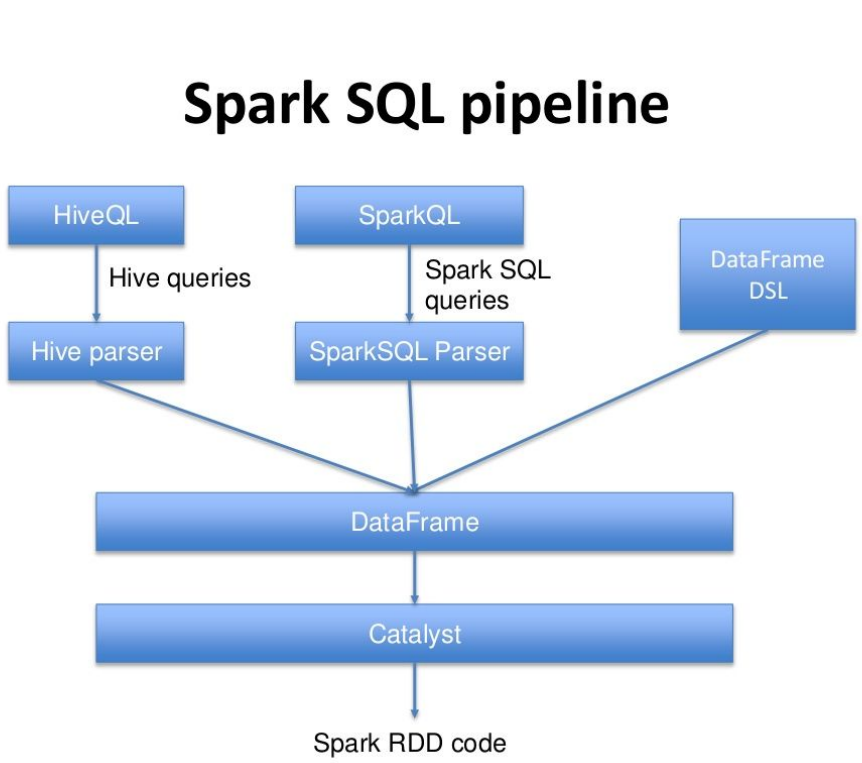




3 ways to analyze the Data Frame :

1. DSL(Domain Specific Language)
2. SparkSQL
3. HQL

Spark SQL Pipeline :

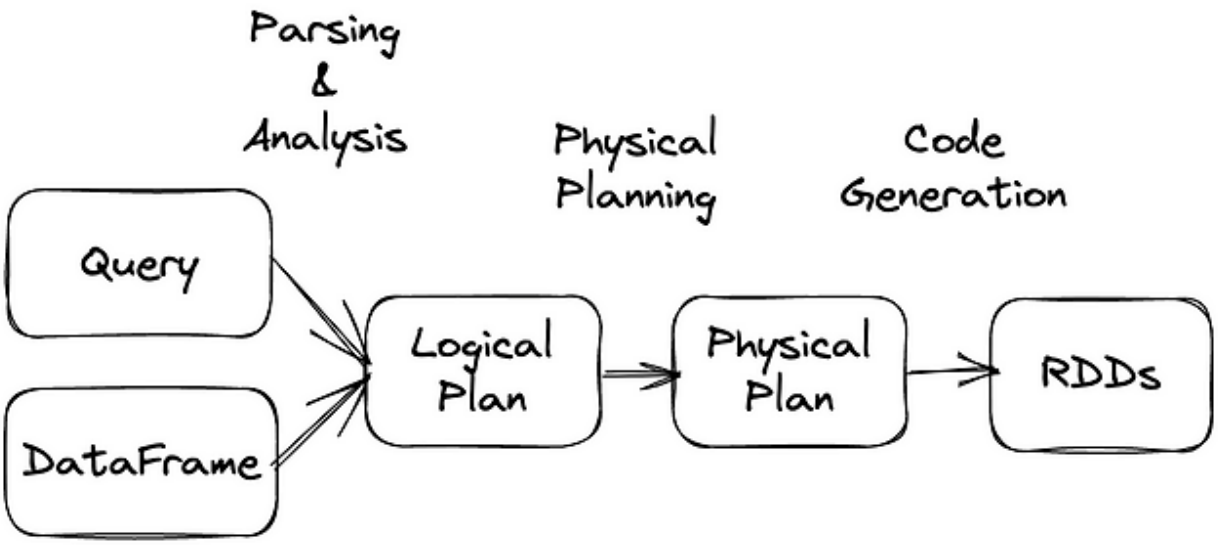


RDD transformations	Data Frame transformation
Actual transformation is shipped on cluster	Optimized generated transformation is shipped on cluster
No schema need to be specified	Mandatory Schema
No parser or optimizer	SQL parser and optimizer
Lowest API on platform	API built on SQL which is intern built on RDD
Don't use smart source capabilities	Make effective use of smart souces
Different performance in different language API's	Same performance across all different languages

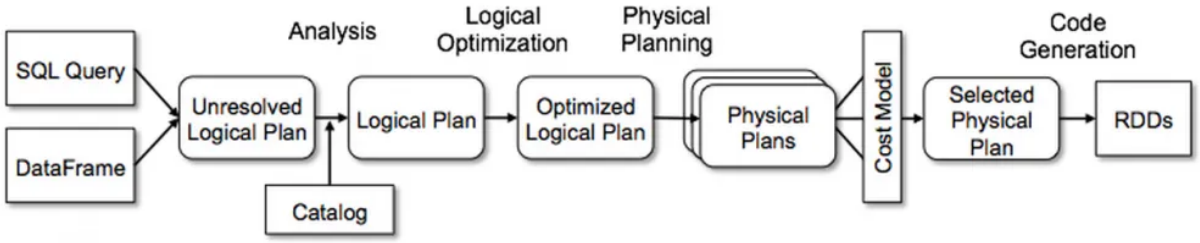
RDD to DF:

1. toDF
2. CreateDataFrame()

Spark SQL execution Plan:



To sum up, it's a set of operations that will be executed from the SQL (or Spark SQL) statement to the DAG which will be sent to Spark Executors.
If you don't know what a DAG is, it stands for "Directed Acyclic Graph". A DAG is an acyclic graph produced by the DAGScheduler in Spark. As a graph, it is composed of vertices and edges that will represent RDDs and operations (transformations and actions) performed on them.
On Spark, the optimizer is named "Catalyst" and can be represented by the schema below. It will produce different types of plans:



Operation names are:

- Analysis
- Logical Optimization
- Physical Planning
- Cost Model Analysis
- Code Generation

Getting various plans

Before Apache Spark 3.0, there was only two modes available to format explain output.

- explain(extended=False) which displayed only the physical plan
- explain(extended=True) which displayed all the plans (logical and physical)

```
x=sql('''select ITEMS.name,
      ITEMS.price,
      SUM(ORDERS.count) as c
      from ITEMS, ORDERS
      where ITEMS.id=ORDERS.itemid
      and ITEMS.id=2
      group by ITEMS.name, ITEMS.price''')
x.explain(extended=true)
```

- explain(mode="simple") which will display the physical plan
- explain(mode="extended") which will display physical and logical plans (like "extended" option)

First step: Unresolved Logical plan generation

This plan is generated after a first check that verifies everything is correct on the syntactic field. Next, the semantic analysis is executed and will produced a first version of a logical plan where relation name and columns are not specifically resolved. This produced this kind of result:

Next step: [Analyzed] Logical plan generation

When the unresolved plan has been generated, it will resolve everything that is not resolved yet by accessing an internal Spark structure mentioned as "Catalog" in the previous schema. In this catalog, which can be assimilated to a metastore, a semantic analysis will be produced to verify data structures, schemas, types etc. and if everything goes well, the plan is marked as "Analyzed Logical Plan"

Next stop: Optimized logical Plan

Once the Logical plan has been produced, it will be optimized based on various rules applied on logical operations (But you have already noticed that all these operations were logical ones: filters, aggregation etc.).
These logical operations will be reordered to optimize the logical plan. When the optimization ends

Then, many physical plans are generated and finally a unique physical plan is selected

From the optimized logical plan, a plan that describes how it will be physically executed on the cluster will be generated. But before selecting a physical plan, the Catalyst Optimizer will generate many physical plans based on various strategies. Each physical plan will be estimated based on execution time and resource consumption projection and only one plan will be selected to be executed.
Based on our example, the selected physical plan is this one (which is the one that is printed when you use explain() with default parameters)

While processing your query, Spark will go through multiple stages. Logical planning will happen first - at this point the query will be parsed, analyzed and optimized, producing "Optimized Plan". It will go into "Physical planning" then, producing an Executed plan, that will be later passed to DAG Scheduler, split into stages, then into tasks, and executed.

Logical plan is a high-level description of what needs to be done, but not how to do it. It has relational operators (Filter / Join) with respective expressions (column transformations, filter / join conditions)

Physical Plan is a much more low-level description. It no longer uses DataFrames, but switches to RDDs instead, and has instructions on what exactly needs to be done, like sorting using specific algorithms etc. For instance, if Logical Plan has just Join node in it - Physical Plan would decide on specific strategy to do the join - for instance, BroadcastHashJoin or SortMergeJoin.

Spark ETL pipeline (Pyspark ETL Pipelines)

Spark Streaming:

Real time processing on Apache Spark

Micro batch:

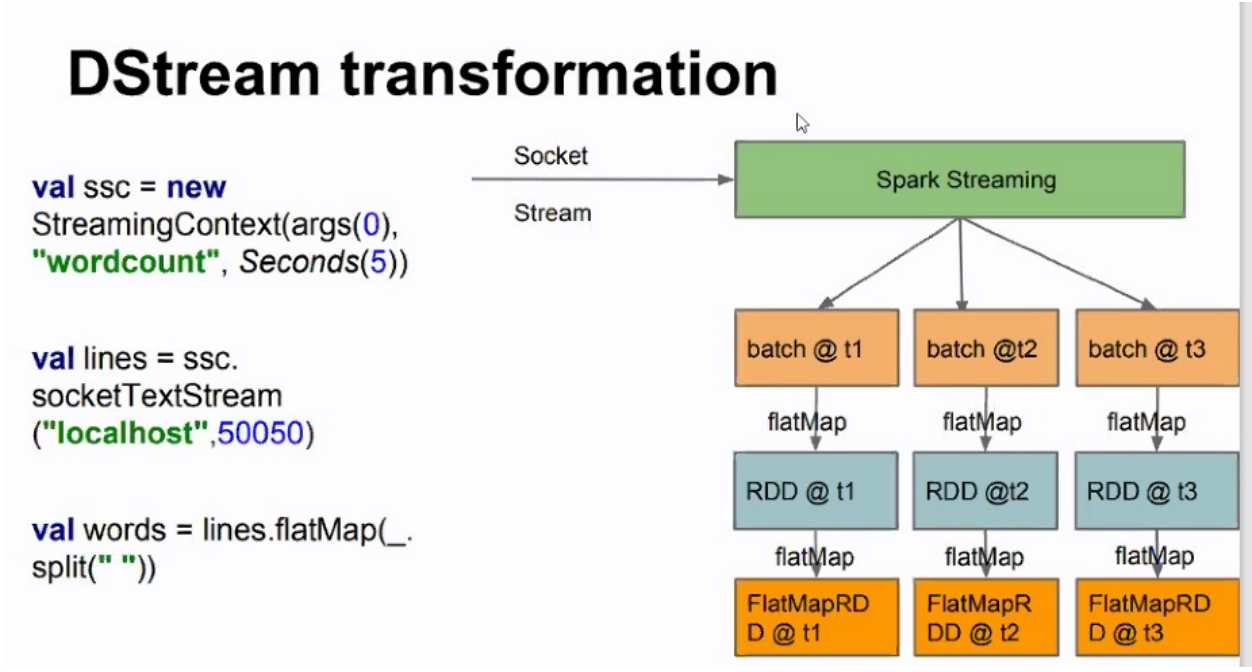
- Spark Streaming is fast batch processing system
- Spark streaming collects stream data into small batch and runs batch processing on it
- Batch can be as small as 1s to as big as multiple horse (these is called ***sliding windows***)
- Spark job creation and execution overhead is so low it can do all that under a sec
- These batches are called as DStreams

- Uses NO SQL DB(Hbase MongoDB , Cassandra) so that BI can connect directly to data and create visualization
- For unstructured live data we use RDD as data from server is stored in memory
- Create cache/disk in separate location for the job if it fails(so that the data will be available for later)

Structured Streaming - when semi structured or structured data is coming and being processed in Data frame then it is called structured streaming

DStream:

1. Discretized stream
2. Each batch of data is converted to small discrete batched
3. Batch size can be constructed from 1s - multiple mins
4. DS can be constructed from
    - Sockets
    - Kafka
    - Hdfs
    - Custom receivers



CMD ->C:\Program Files (x86)\Nmap>ncat -l 9999 (for windows, go to location and start cmd)

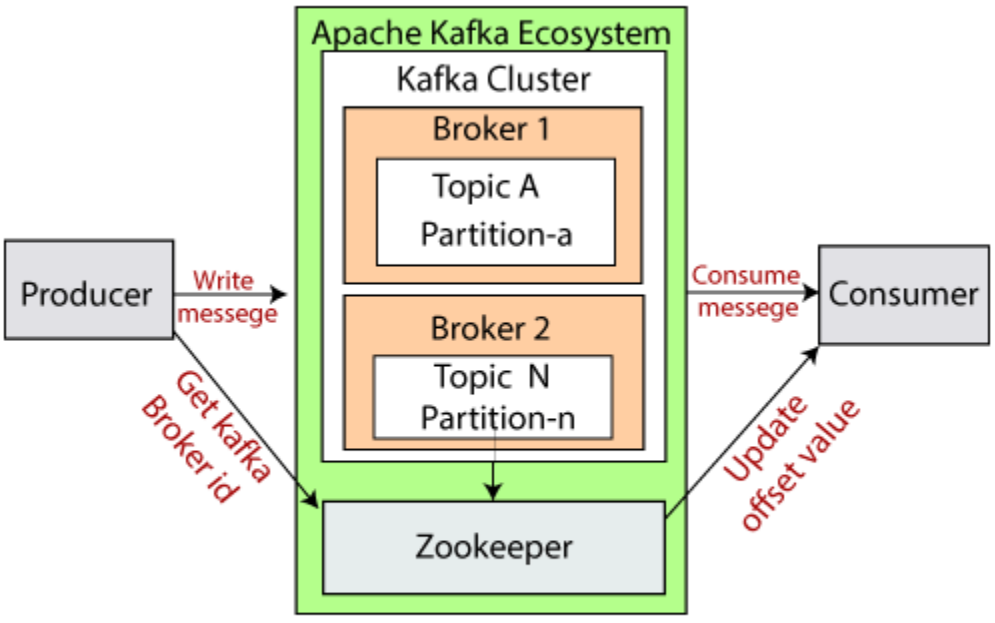
To do spark-submit app.py on cloudera (app.py will contain following code)
we can run following code directly in jupyter as well on windows

from pyspark import SparkContext

```
from pyspark.streaming import StreamingContext

sc = SparkContext('local[2]','StreamingWorldCount')
ssc = StreamingContext(sc,3)

lines  = ssc.socketTextStream("localhost",9999)

words = lines.flatMap(lambda line : line.split())
pairs = words.map(lambda word : (word,1))
wordCounts = pairs.reduceByKey(lambda x,y:x+y)

wordCounts.pprint()

ssc.start()
ssc.awaitTermination()
```

Apache Kafka:

Architecture :



Apache Kafka Architecture

- Single topic can have multiple consumers
- For each consumer we need separate partition of the topic
- Single partition will allow only one consumer to read (That means number of consumer = number of partitions of a topic)

Use of Zookeeper in KAFKA world:

Zookeeper is used for metadata management in the Kafka world. For example: Zookeeper keeps track of which brokers are part of the Kafka cluster. Zookeeper is used by Kafka brokers to determine which broker is the leader of a given partition and topic and perform leader elections.

When consumers belong to the same consumer group, each of them reads from one or more partitions of the topic. Remember the each partition can be read from only one consumer. So for example, topic with 4 partitions and you have 2 consumers, each of them will read messages from 2 partitions; if you go up to 4 consumers each will get 1 partition; if you go to 5 consumers, the last one will be idle because no partitions are available. If you have consumers in different consumer groups they will receive same messages as a "broadcast"
Stackoverflow -> in-kafka-is-each-message-replicated-across-all-partitions-of-a-topic
Replication does not occur across partitions. Each message goes into a single partition of the topic, no matter how many partitions the topic has.

If you have set the replication-factor for the topic to a number larger than 1 (assuming you have multiple brokers running in the cluster), then each partition of the topic is replicated across those brokers.

Z-Ordering

Z-order on most queried, high-cardinality columns.

- Use Z-order indexes alongside partitions to speed up queries on large datasets.
- Z-order clustering only occurs within a partition, and cannot be applied to fields used for partitioning.

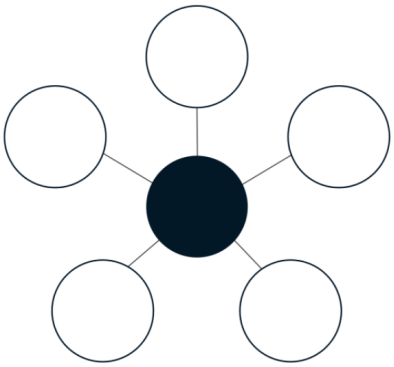
- Liquid Partitioning

- MapSide Join , reduce side join in hadoop ??
- Mapside join : no data shuffling(small data will be cached)(Map-side join is a method of joining two datasets in PySpark where one dataset is broadcast to all executors)
  Reduce side join : It Default join in which shuffling happens

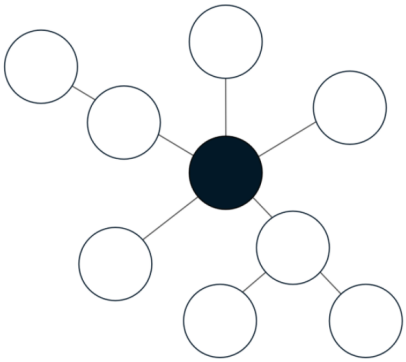
SnowFlake :

- It's a Data WareHouse
- Structured , un structured , semistructured data
- Multi Clustered distributed architecture
- Internal stage and external stage(to stage local and other source data)
- Security , schema , low cost , multi clustered distributed architecture ,we can do everything in snow flake (visualization, ingestion , analysis , storage), user portal
- We can connect with Any cloud storage , 3rd party BI tools
- Provides SQL and Python API's
- With STAGE we can load 'n' MB of data into snowflakes
- 'Copy into' and Snowpipe - ways to get data

Star schema



Snowflake schema



Data models :

1. Star Schema
2. SnowFlake Schema

Skew Serialization

Spill @ Shuffle @ storage.

- **DFS**
    1. Data Lake
    2. HDPS + YARN
    3. Unlimited storage any type of data , structured, un-structure , semi-structure

- **Databases (OLTP)**
    1. RDBMS
    2. SQL, MySQL
    3. Limited storage, Structured data
    4. Row oriented

- **Big Data Databases**
    1. Large amount of structured & semi structured data
    2. NoSQL, HBase, Mongodb, Cassandra, CouchDB

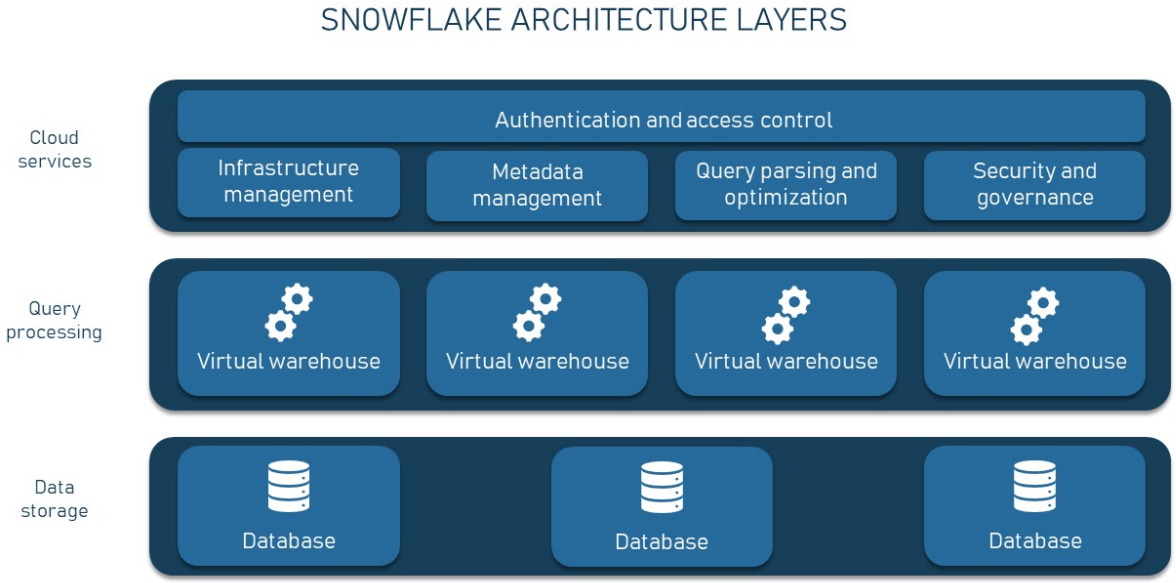
- **Data Warehouse**
    1. Large amount of structured & semi-structured date
    2. Oracle, Redshift (AWS), Synapse (Azure) , Hive
    3. SNOWFLAKE
    4. Schema and ‘column oriented’
    5. OLAP

- **Object Storage**
    1. S3(AWS), Azure storage.

- **Delta Lake**
    1. Structured, semi structured & Unstructured - large amount -100% ACID transactions , Available in Databricks

LakeHouse
- Combination of DBFS , Storage , Spark

SnowFlake Architecture :



Star Schema :
- Storage : More Storage Required(due to denormalized tables)
- Design : easy design

- Scalability : High scalability
- Efficiency : more efficient
- Small warehouse

Snowflake Schema :

- Storage : less storage (highly normalized tables)
- Design : complex design
- Scalability : Low Scalability
- Efficient : less efficient
- Warehouse : Large warehouse

SnowFake Terminologies :

- Internal stage
- External stage
- Data sharing
- Snow pipe
- Put - command
- Integration API
- Snow sight
- Query acceleration service
- Optimized wareHouse
- Now Flake Connector
- Result Cache
- Time travel (90 days time travel)
- SnowSQL
- Copy - Statement
- Failed-Safe

Note - By default single warehouse is a single cluster or Maximum clusters –10

Warehouse Size

Size specifies the amount of compute resources available per cluster in a warehouse. Snowflake supports the following warehouse sizes:

Warehouse Size	Credits / Hour	Credits / Second	Notes
X-Small	1	0.0003	Default size for warehouses created using CREATE WAREHOUSE.
Small	2	0.0006	
Medium	4	0.0011	
Large	8	0.0022	
X-Large	16	0.0044	Default for warehouses created in the web interface.
2X-Large	32	0.0089	
3X-Large	64	0.0178	
4X-Large	128	0.0356	
5X-Large	256	0.0711	Generally available in Amazon Web Services regions, and in preview in US Government and Azure regions.
6X-Large	512	0.1422	Generally available in Amazon Web Services regions, and in preview in US Government and Azure regions.

- What is internal stage
- Copy into command

How to connect Command line on local with snow fake ;

1) Download SnowSQL - Snowflake Developers
2) https://app.snowflake.com/onokpow/sb11405/w5HMlKsnKzOF#query user url after login to snowflake on web
3) Run into CMD to connect snowsql -a onokpow-sb11405 -u farhan
4) Connect with database.schema

Stages in snowflake:

1. Internal - to load local storage

2. External- to load data from external storage like AWS , Azure

Data models - star schema and snowflake schema

Snowflake vs data bricks -

- Snow flake is Shared disk architecture

COPY INTO: Data from stage to table

Can load data from local to snowflake
Steps to load data:

1. Connect Snow SQL CLI
2. Create stage (internal stage) - > create stage my_internal_stage; show stages; ->to see available stages
3. Create empty table in data warehouse ->
4. Put data into stage -> file://C:\Users\Futurense\Downloads\Retail_Data.csv @my_internal_stage;
5. Run “Copy into” in internal stage -> copy into customers from @my_internal_stage/Retail_Data.csv file_format=(type='csv',skip_header=1);

‘List’ command to see content inside Stage -> list @my_internal_stage;

External Stage:

```
create or replace schema external_stage;

CREATE OR REPLACE STAGE NEW_DB.external_stages.aws_stage url =
's3://bucketsnowflakes3';
Or
CREATE OR REPLACE STAGE MANAGE_DB.external_stages.aws_stage url =
's3://bucketsnowflakes3' credentials = (aws_key_id = 'ABCD_DUMMY_ID' aws_secret_key =
'1234abcd_key');
Or
CREATE OR REPLACE STAGE aws_stage url = 's3://bucketsnowflakes3';
```

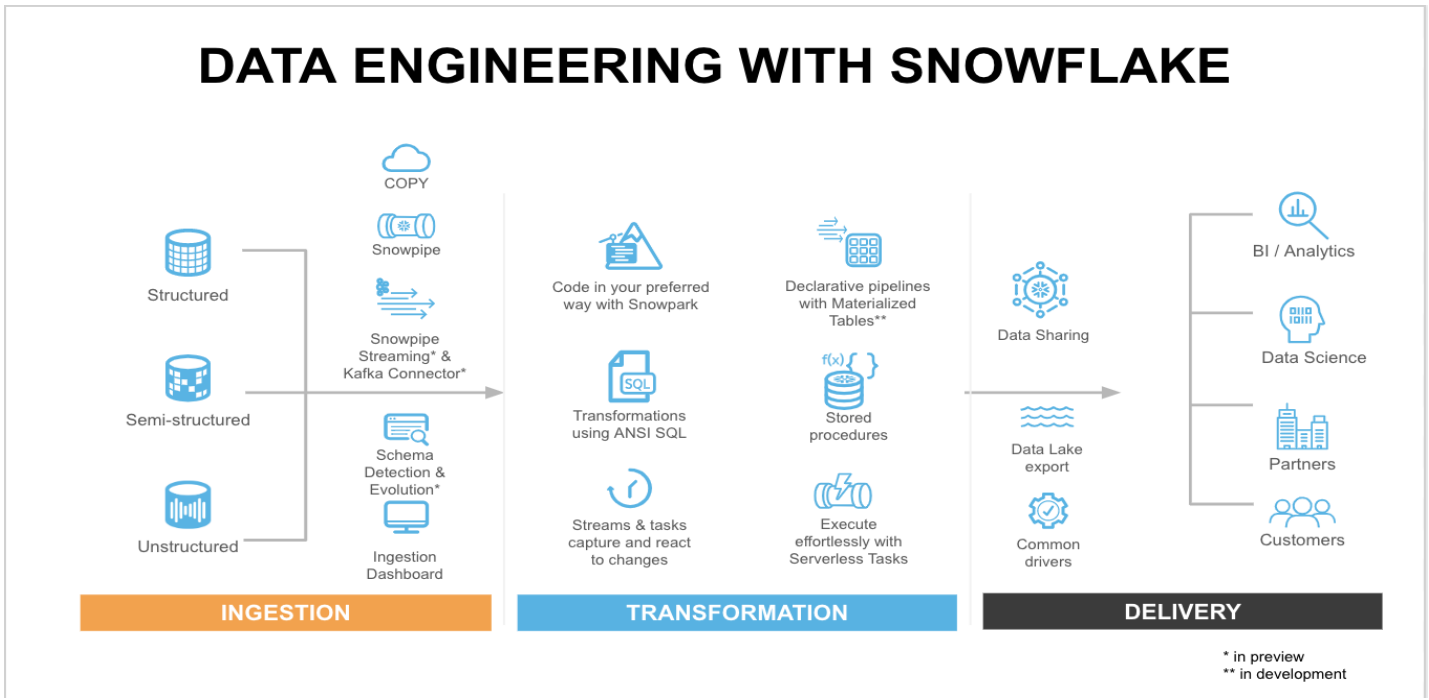
Aws

```
create or replace file format NEW_DB.EXTERNAL_STAGE.my_csv_format
                                        type = csv
                                        field_delimiter = ','
                                        skip_header = 1
                                        null_if = ('NULL', 'null')
                                        empty_field_as_null = true;
create or replace storage integration s3_int
  type = external_stage
  storage_provider = s3
  enabled = true
  storage_aws_role_arn = 'arn:aws:iam::530595819296:role/snowflakerole'
  storage_allowed_locations = ('s3://newproductbucketsk/');
```

Snow_pipe :
Is a serverless data ingestion service that automates the loading data from different sources
S3 , azure storage, google storage (automate data loading)
Copy into : manual loading

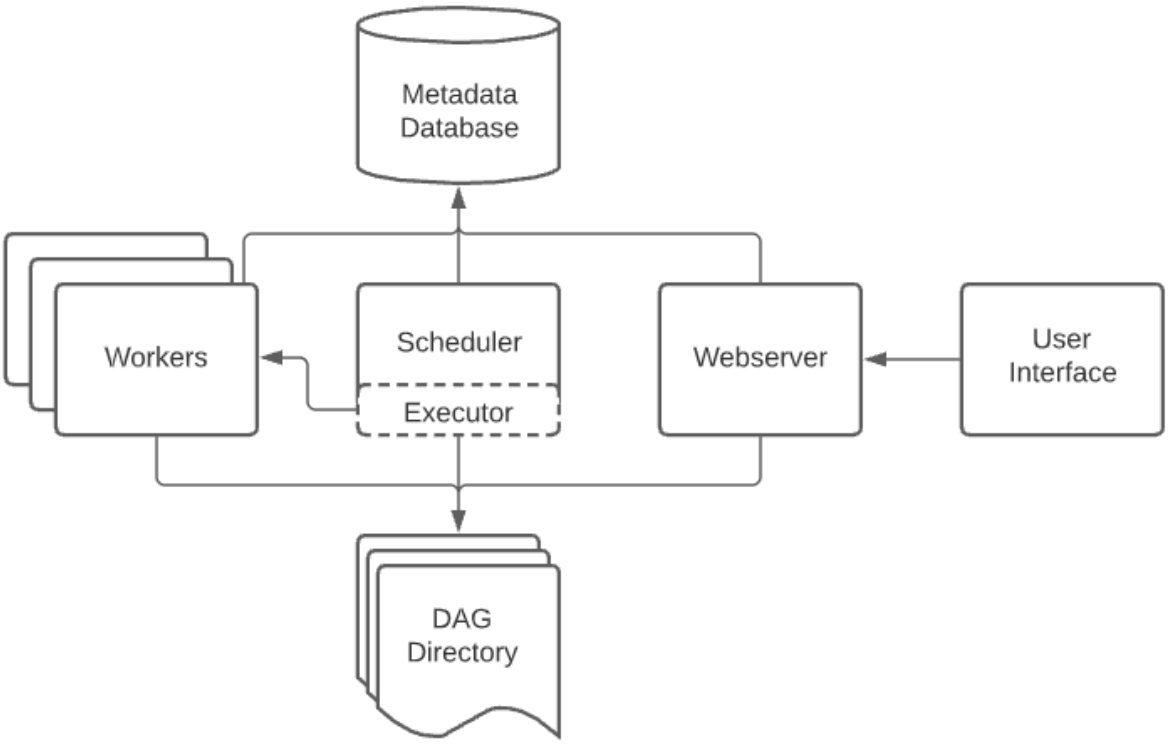
'Result cache' in snowflake - > using to store query result
Stores data as **micro partitions**

AWS S3 bucket -> we can use 'Athena' to use query language to analyze S3 files



AirFlow :

Airflow Architecture:



- Airflow is a platform to built and run workflows
- A workflow is represented as a dag
- Dag contains individual pieces of work called Task , arranged with dependencies and data flow
- A components of airflow architectures
    - Web Server(GUI)
    - Scheduler
    - Executor - to execute the task
- Airflow will play in Single node and Multinode cluster - Types of Architecture

How to create the WorkFlow (DAG):

1. Make the Imports : the first step is to import the classes you need.
2. Create the WorkFlow DAG object, after having made the imports, the second step is to create the Airflow DAG object
3. Add your Task
4. Defining Dependencies

LegatoAI Mock Interview:

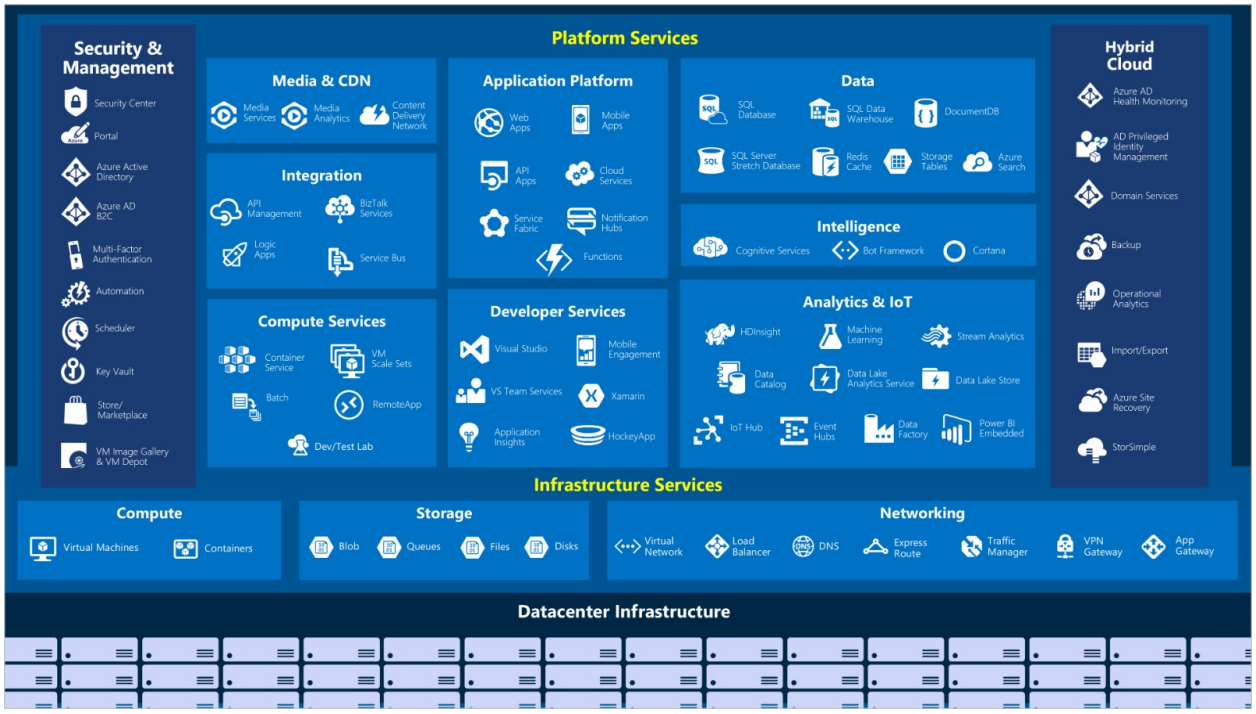
- Duplicate vs duplicated
- Types of Window Functions - Analytical, ranking , Aggregation

- Frame cluse
- Pivotes
- Optimisations in Spark
- ntile

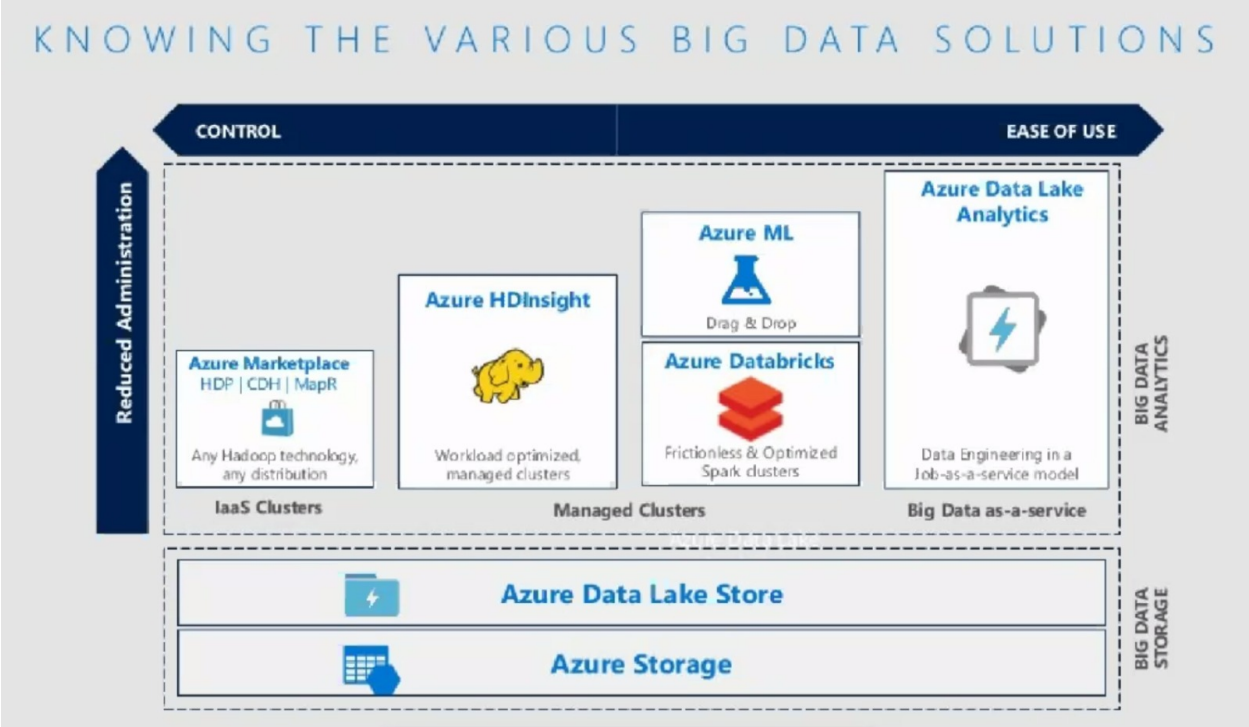
Azure :

Types of ADLS accounts : GEn1 and GEN2
Data warehouse in azure - Synapse

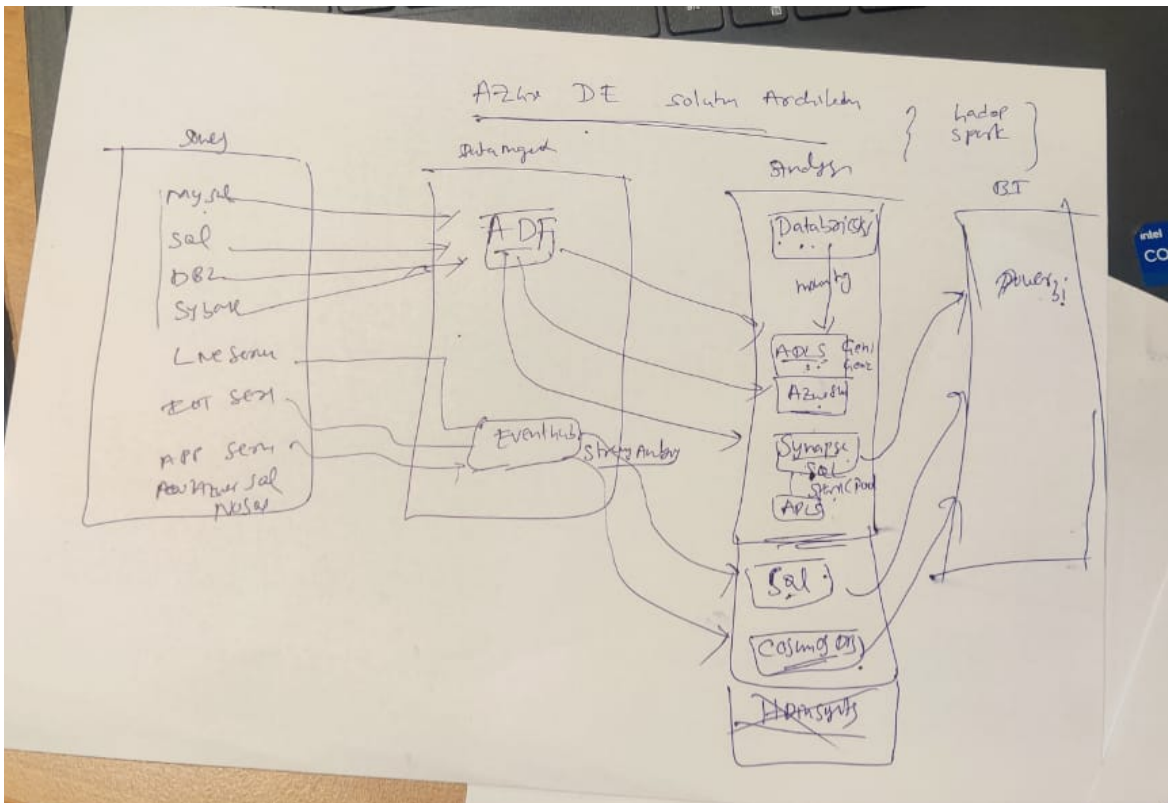
IASS : Virtual machine with processor, ram, cores, network , VPN
Pass : HDinsight (CLI based),DataBricks(GUI) , Synapse(Data warehouse), Azure SQL , CosmosDB(NoSQL DB)



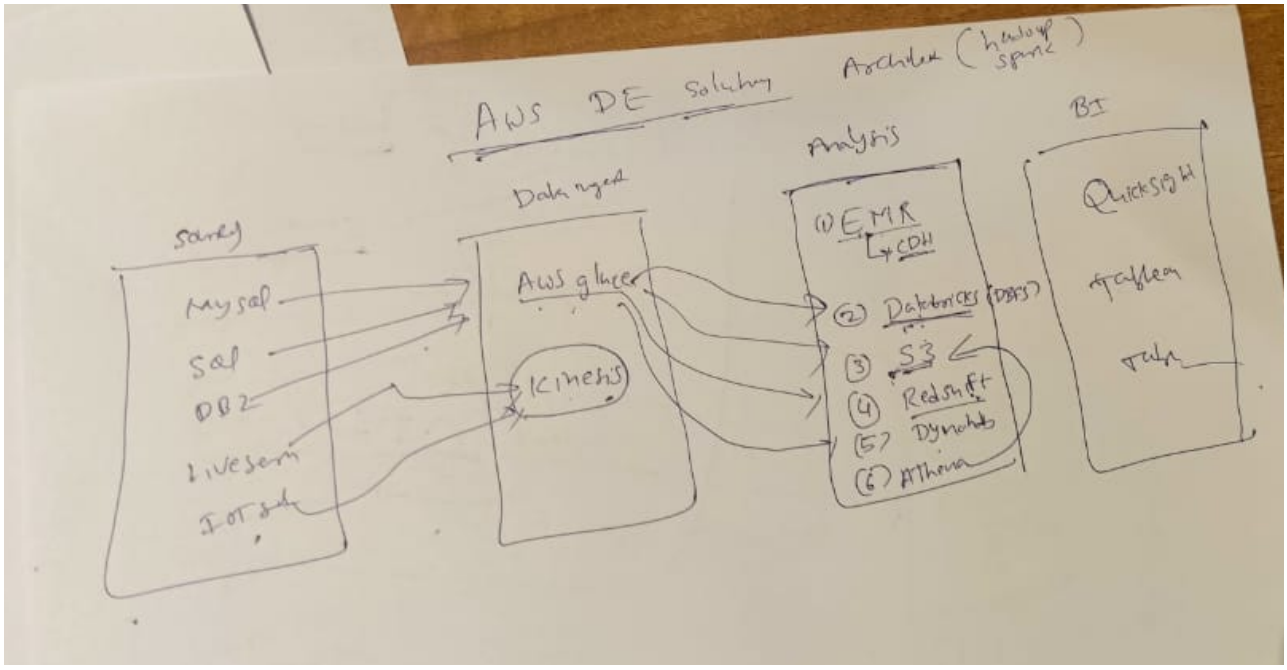
ADF : ETL tool(create the pipelines)



Azure Architecture:



AWS architecture:



Azure storage :

- 100 storage accounts for single subscription
- 500TB per storage account
- A Storage account is uniquely addressable
- Available from anywhere using REST API

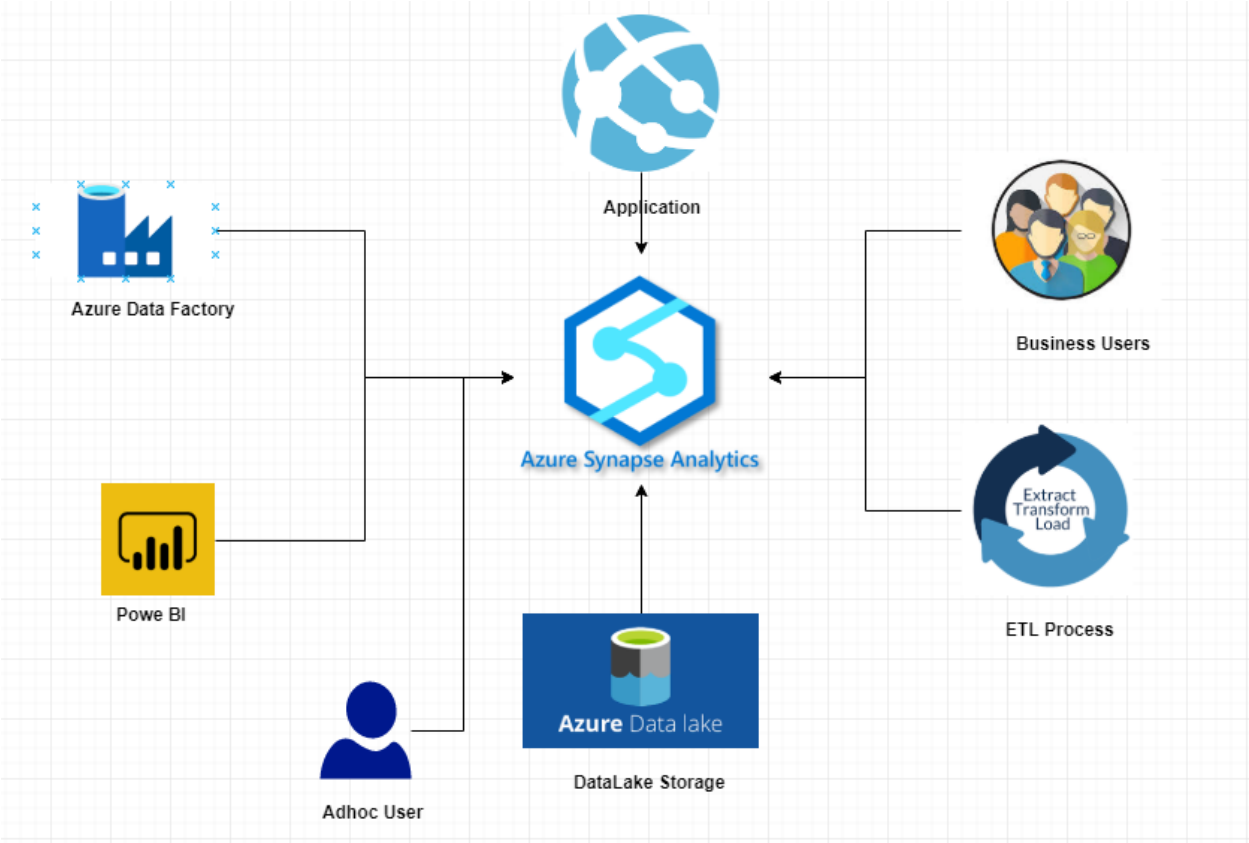
Azure Synapse :

Data warehouse

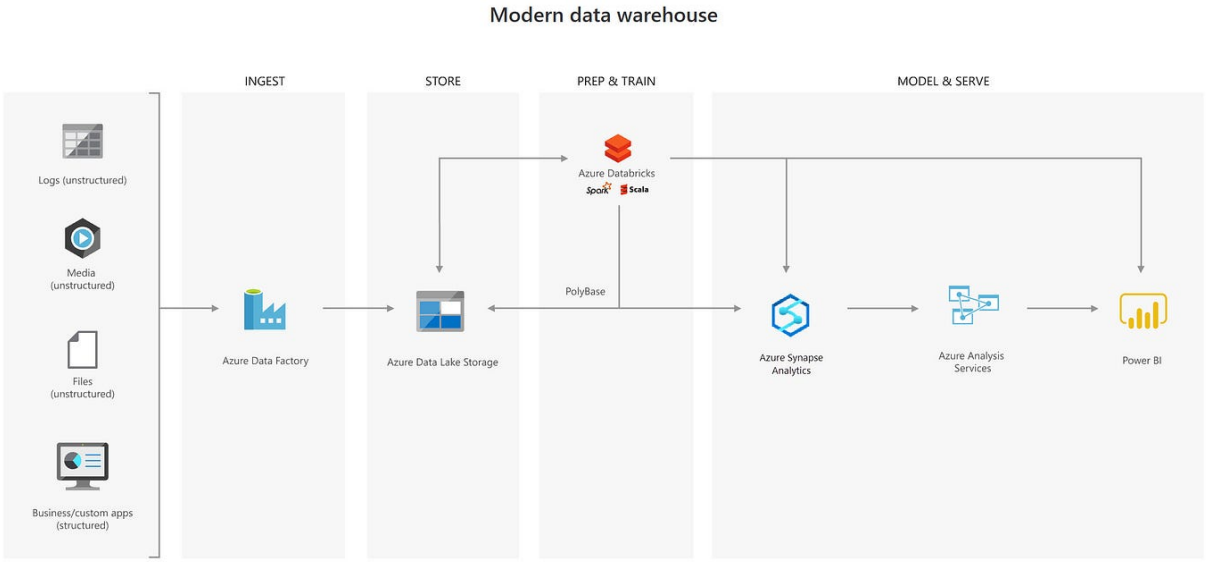
- Two libraries available
    - SQL
    - Spark Pool

Azure Synapse is an enterprise analytics service that accelerates time to insight across data warehouses and big data systems. Azure Synapse brings together the best of SQL technologies used in enterprise data warehousing, Spark technologies used for big data, Data Explorer for

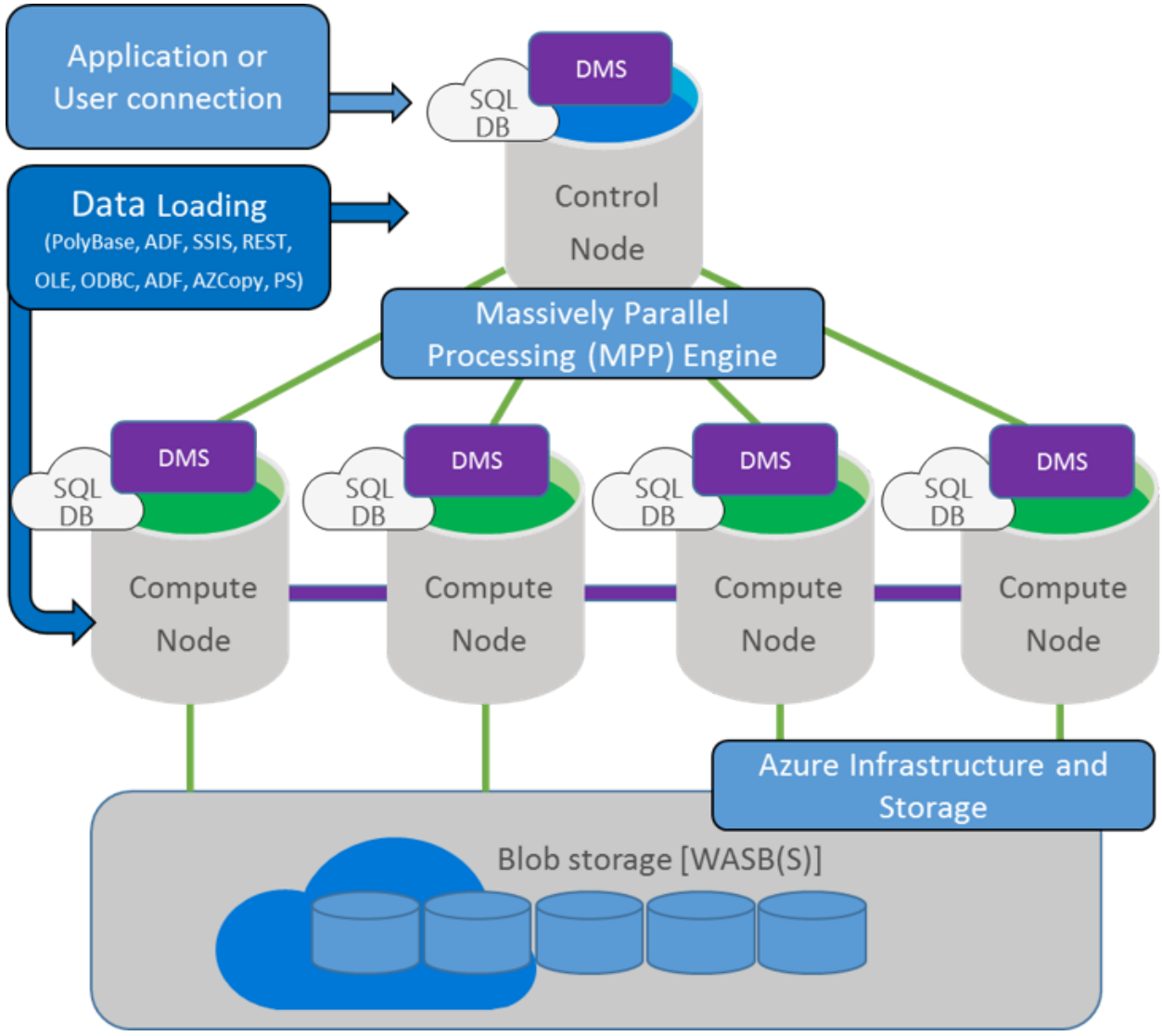
log and time series analytics, Pipelines for data integration and ETL/ELT, and deep integration with other Azure services such as Power BI, CosmosDB, and AzureML.



Synapse Data engineering Solution Architecture:



Azure Synapse Architecture:

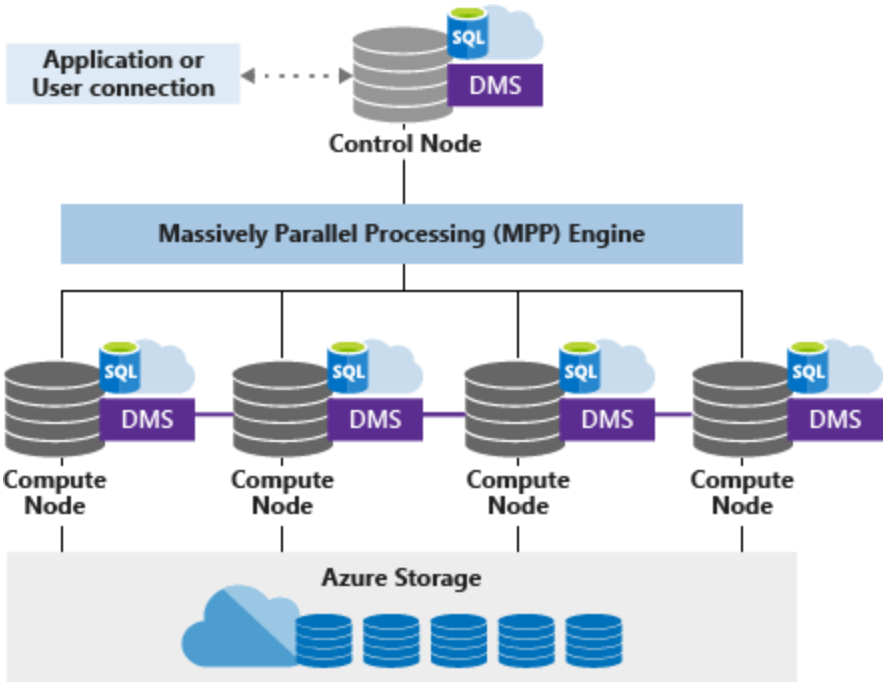


Synapse components :

- Workspace

Synapse SQL architecture components

Dedicated SQL pool (formerly SQL DW) leverages a scale-out architecture to distribute computational processing of data across multiple nodes. The unit of scale is an abstraction of compute power that is known as a data warehouse unit. Compute is separate from storage, which enables you to scale compute independently of the data in your system.



Dedicated SQL pool (formerly SQL DW) uses a node-based architecture. Applications connect and issue T-SQL commands to a Control node. The Control node hosts the distributed query engine, which optimizes queries for parallel processing, and then passes operations to Compute nodes to do their work in parallel.

The Compute nodes store all user data in Azure Storage and run the parallel queries. The Data Movement Service (DMS) is a system-level internal service that moves data across the nodes as necessary to run queries in parallel and return accurate results.

With decoupled storage and compute, when using a dedicated SQL pool (formerly SQL DW) one can:

- Independently size compute power irrespective of your storage needs.
- Grow or shrink compute power, within a dedicated SQL pool (formerly SQL DW), without moving data.
- Pause compute capacity while leaving data intact, so you only pay for storage.
- Resume compute capacity during operational hours.

Azure Storage

Dedicated SQL pool SQL (formerly SQL DW) leverages Azure Storage to keep your user data safe. Since your data is stored and managed by Azure Storage, there is a separate charge for your storage consumption. The data is sharded into distributions to optimize the performance of the system. You can choose which sharding pattern to use to distribute the data when you define the table. These sharding patterns are supported:

- Hash
- Round Robin
- Replicate

Control node

The Control node is the brain of the architecture. It is the front end that interacts with all applications and connections. The distributed query engine runs on the Control node to optimize and coordinate parallel queries. When you submit a T-SQL query, the Control node transforms it into queries that run against each distribution in parallel.

Compute nodes

The Compute nodes provide the computational power. Distributions map to Compute nodes for processing. As you pay for more compute resources, distributions are remapped to available Compute nodes. The number of compute nodes ranges from 1 to 60, and is determined by the service level for Synapse SQL.
Each Compute node has a node ID that is visible in system views. You can see the Compute node ID by looking for the node_id column in system views whose names begin with sys.pdw_nodes. For a list of these system views, see Synapse SQL system views.

Data Movement Service

Data Movement Service (DMS) is the data transport technology that coordinates data movement between the Compute nodes. Some queries require data movement to ensure the parallel queries return accurate results. When data movement is required, DMS ensures the right data gets to the right location.

Distributions

A distribution is the basic unit of storage and processing for parallel queries that run on distributed data. When Synapse SQL runs a query, the work is divided into 60 smaller queries that run in parallel.

Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. A dedicated SQL pool (formerly SQL DW) with maximum compute resources has one distribution per Compute node. A dedicated SQL pool (formerly SQL DW) with minimum compute resources has all the distributions on one compute node.

Synapse vs Databricks

Synapse	Databricks

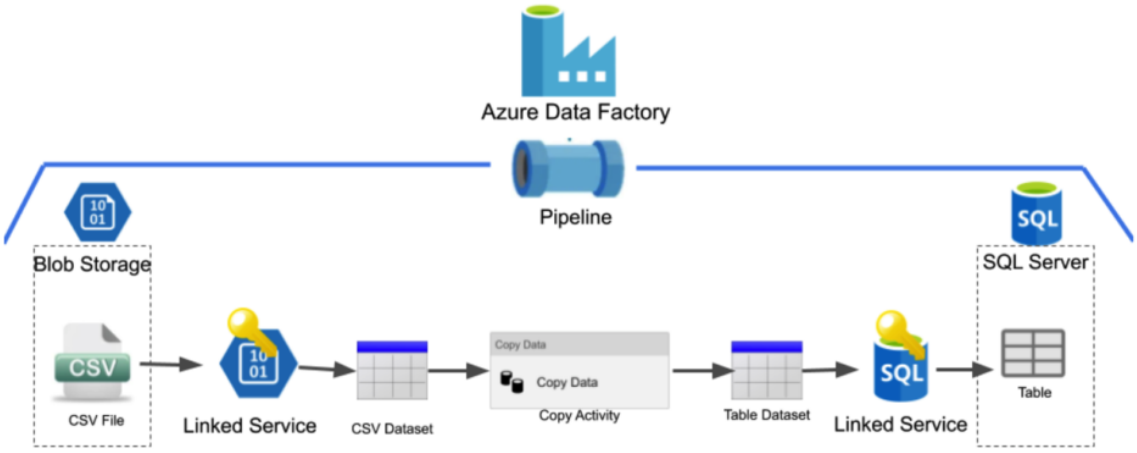
No cluster management	Cluster management
Auto cluster pause	No pause
More cost	Less cost
Auto scaling	Auto scaling
Spark platform and sql platform	Pure spark platform

- What is control node in synapse
- Compute node?
- What is synapse
- Synapse component
- Linked service
- Components used to get data into ADLD with synapse - linked service and integration services
- What is the method to create pipeline in synapse - > go to integrate services create link connection and then create a pipeline
- Types of activities in synapse - copy data and transform data

Azure Data Factory:

What is inside the azure data factory?:

- Activities
- Linked services
- Datasets
- Data flow
- Triggers
- templates



Steps to create data factory pipeline:

- **Copy to** - For single file transfer
- **ForEach** - to transfer multiple files with single pipeline

Steps - > Properties -> Source data store->Configuration->Destination->Settings

Properties :

1. Task type:
    a. Built In copy tasks(90+ sources)
2. Task cadence or task schedule:
    a. Run once now
    b. Schedule
    c. Tumbling window

Source data store:

1. Source:
    a. From where the data is coming
2. Connection:
    a. Linked service(dataset)
3. File or folder:
    a. Select Folder/file path
4. Filter by last modified:
    a. Start time
    b. End time

Configuration:

1. Preview data , change delimiter , end line parameter , data type

Destination:

1. Select target source linked service
2. Select target folder
3. change file name
4. File format setting
    a. Delimiter column row

Settings:

1. Task name
2. Description
3. Fault tolerance

Hands On

- Scenario 1:
    - Create the data factory pipeline from the source blob to blob
- Scenario 2
    - ADLS to SQL

Activities in ADF :

There are conditional activities in ADF

1. **IF condition activity:**
    a. Used to specify expression that evaluates the true or false in specific activities once expression is evaluated then corresponding activity will be executed
2. **Switch Activity:**
    a. It is also used to specify an expression that evaluates to a string value and up to 25 corresponding activities

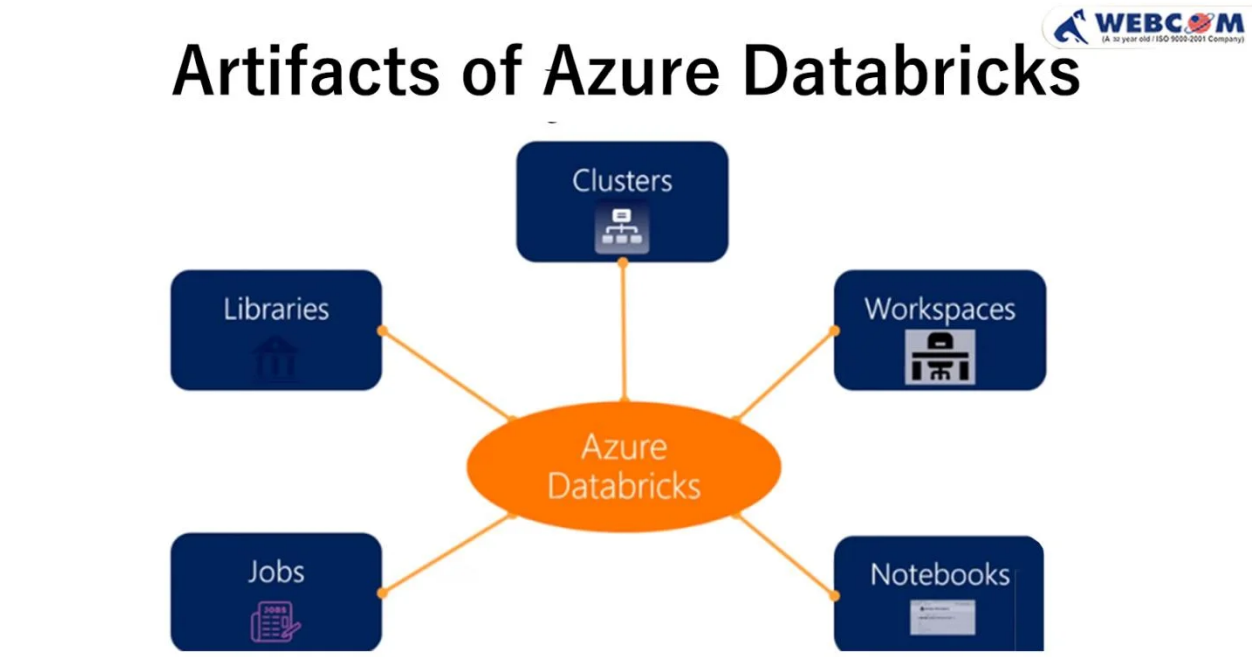
Azure key vault:

- Used to secure the repository for secrets and cryptography keys , secrets means name stored in azure key vault

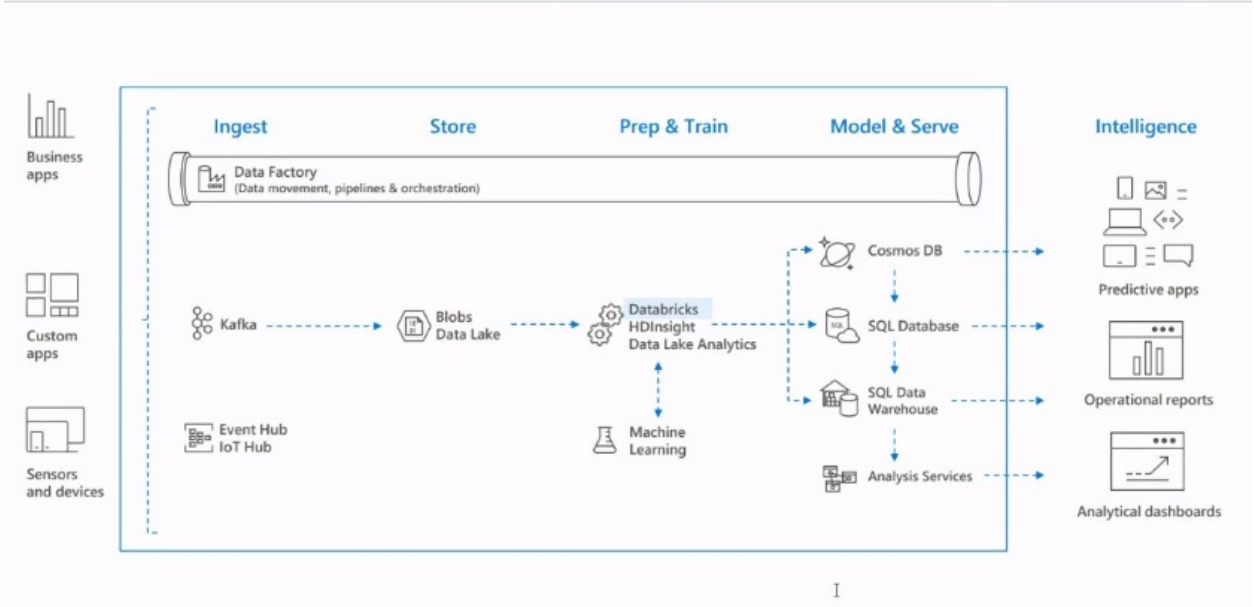
Service Principal :

- It is an identity created to use with application or service , ex - DataFactory
- Service principal use to identify the services when external resources require authentication and authorisation

DataBricks:



Engineering Architecture diagram of Databricks:



Types of clusters in Azure Databricks:

Azure Databricks supports three cluster modes:

1. Standard
2. High Concurrency
3. Single Node