**-> BIG DATA**

* Describes large amount of data, both structured and unstructured
* 5V’s :
  + Volume : incredible amount of data generated each sec from different sources
  + Velocity : speed at which new data is generated and speed at which it moves around
  + Variety : data is present in diff forms , structured, semi structured, unstructured
  + Veracity : quality or trustworthiness of data
  + Value : until a data is converted to a valuable form is useless, thus value is important
* Issues with Big data :
  + Management issues : security , complexity
  + Storage issues : storage and overall cost,
  + Processing issues : processing , consumes a lot of time
* Hadoop is used for batch processing of data, there is a time lag
* Spark is used for processing real time data, no time lag in processing

**-> APACHE SPARK**

* Cluster computing framework for real time processing
* Features:
  + Real time computation , low latency , 100times faster for large scale data processing,
  + Polyglot : we can write spark app in multiple languages
  + Simple prog layer provides powerful caching, disk persistence
  + Deployment :
* Hadoop posed difficulty in predicting,
* Spark plus hadoop allows processing and analytical tasks at a speed of 40ms per event
* Most preferred lang with spark is scala
* Spark uses master worker architecture
* Master has a driver prgram which drives the application
* Here spark context is created and its the gateway to all the functionalities
* Driver program and spark context take care of the job execution in the cluster
* Spark context also works along with cluster manager, which is responsible for acquiring resources for the cluster to perform the job
* This job is divided among the worker nodes to perform it and send back the results to spark context
* There is executor in worker node which is distributed process which is responsible for running the task
* For each spark app there is an executor and remains for the entire lifetime of the app
* Spark Cluster manager types
* In memory data sharing is lot faster than network and disk sharing
* RDDs in Spark : Resilient Distributed Dataset ,
  + Resilient : fault tolerant , can recover the missing or lost partition in case of node failure with the help of rdd lineage graph
  + Distributed : since data resides on multiple nodes
  + Data set : data record
  + There are 3 ways to create an rdd
  + Parallelize a collection
  + From external source of data
  + From existing rdd
* Operations :
  + Transformation : create new rdd from existing rdd, here we can take “**map**” function and do some operations on the rdd, here whatever we want to do is done on each element of rdd, “**flat map**” , diff bw **flat map** and **map** is that “**flat map**” can return a list of elements, where “**map**” returns only 1 element, “**filter**” will return only those elements which satisfy the predicate, “**intersection**” to get common elements bw 2 rdds but both rdds should be of same type
  + Actions : for working with actual data set we use them but new rdds are not created and their result is stored in drivers or to the external storage system, “**reduce**” to add each element of rdd, or concatenating the elements, it produces the output of the same type, “**first**” gives the first element of rdd, “**take**” to get starting amount of elements(like first 4), “**foreachpartition**” to perform operations on elements of each partition
* Components :
  + Spark SQL : module for structured data processing, code can be written in any lang, helps to query data from data source, can mix sql like queries with spark propgrams using api’s, provides unified data access thus provides single interface to interact with diff types of structured data, performs hive queries on data, std connect through jdbc,odbc, also scalable,
    - toDF() = used to convert a list to data frames
    - show() = to show the data frame
    - printSchema() = tells the schema of data frame
    - select() = to select data from data frame
    - filter() = to issue a predicate
    - groupBY() = as in sql
    - df.createOrReplaceTempView() = to get a view of df
* Json data falls in semi structured data
* Spark .read.json(“file name”) to read json data and will create a data frame for it
* Spark supports reading xml file but it may not work in some scenarios
* Use spark xml databricks for handling xml data
* If we want to put a file in linux to hadoop, we use data ingestion tools like kafka, kinesis etc
* There is file format called AVRO and is used to serialize ur files, and compress the data,
* To handle AVRO data we will use AVRO reader
* To handle it we can use github to get reader for it
* If any file is not readable using spark we have to use some parser or write our own parser
* When we are talking about compression techniques in big data there are 3 formats
  + RC : 3rd best , very old
  + ORC : best compression format (optimized row columnar format), if hortonworks is used this is used, is able to do indexing internally
  + PARQUET : 2nd best, if cloudera is used this is used, achieves 50% compression ,
* we cant read parquet files directly with text editors
* Spark.read.parquet() is used to read parquet files
* To handle databases like sql databases, for this we need a driver to connect to that database, like jdbc or odbc, ie mysql connector.
* To read data from a database, we need to instantiate an object of java.utils.Properties class because this object has some attributes using which we can establish connection with rdbms , attributes like database name, table name etc.
* And then put the username,password, driver class, url, table name to connect in the object to connect to the database
* Spark.read.jdbc() with url , table, object is used to fetch and read the data from the database
* MySQL is faster than Spark
* We use spark to read from SQL database because to perform complex operations on complex data
* Hive is a data warehouse architecture which provides interaction between user and hdfs
* Hdfs = Hadoop Distributed File System and is the primary storage used by Hadoop Applications
* To integrate hive and spark copy the hive-site.xml to the spark configuration folder(spark/conf)
* we initiate the spark shelll first and then execute the commands
* To connect from spark to hive we need to create a hive context
* It is created using creating a new object as “**new org.apache.spark.sql.hive.HiveContext(sc)**”
* we will use this hive context for operations on the hive table
* In latest versions of spark we can access the tables from database and hive directly using spark.sql()
* If we want to store a dataframe back to table either in hive or database using latest version of spark = dataframe\_name.write.mode(SaveMode,Overwrite).saveAsTable(“name of the table”)
* There are 3 write modes : overwrite, append, replace
* When we cache a dataframe the internal rdd will get cached
* We can also uncache it
* Spark.SQL.files.maxPartitionBytes : specify the max number of bytes that will be packed in a partition
* Spark.SQL.files.CostInBytes : whenever we try to open multiple files together, it specifies the total cost to open them scan them , this cost may be in the form of time required to read through all the files
* Spark.SQL.broadcastTimeout
* SparkSQL had a drawback and that is if the partition size if more than 2gb the shuffled operations will fail
* If there are 2 partitions with each size of 5gb and if we try to join these data frames or the partitions an error will occur as shuffle operations are not supported for partitions with size > 2gb
* So in this case we will repartition the data using repartition(number of partitions in which we want to distribute the data)

**-> APACHE KAFKA**

* When we need to process the data in real time we use it
* It is designed to efficiently collect real time data
* It is designed to handle large amount of data and can be integrated with any big data tool or framework.
* There two app in kafka :
  + Producer : which send the data to kafka cluster
  + Consumer :subscribe and access the data from kafka cluster
* It cant transform the data, for it we require a processing engine
* Kafka streams aim to add the processing ability to kafka
* From the point of view of kafka message is nothing more than byte array and so we can store data of any type
* Spark streaming is consumer to kafka cluster
* Spark streaming is real time analytics engine
* Apache flume is used to get the data and send to kafka
* Zookeeper is used to store information about cluster status and consume offsets
* All messages are sent as key value pairs
* Every message on broker gets an offset number
* Writing message to kafka is appending only and there is no modification
* Any data sent to kafka it will be retained for a week
* For each partition kafka will select a broker for leader out of all the brokers which have that partition
* There is cold replication of partitions
* Its the responsibility of consumer to remember till where it has read
* Each consumer stores the offset till it read
* if we are dealing with 100% real time data then we can delete that data once read
* **Zookeeper :** original idea was to maintain status of different services
* We can register with zookeeper as a service and can store the information about the service and ask about the status of the service at any time from it
* Zookeeper internally is a cluster of 3 or 5 machines, thus it is a centralized place where status of service is stored and can be queried
* Zookeeper will hold the metadata of entire kafka cluster
* Jps command will tell u all the java processes running in my system
* Before running the kafka cluster we need to run the zookeeper and by default zookeeper runs on port 2181
* Then on another session start kafka
* First thing after starting a kafka cluster is create a topic
* each consumer will have its own metadata
* When we want to read a time sequence data thus it should be in sequence only then we store the data to only 1 partition and let the consumer to read the data from that partition as reading data from multiple partitions wont be in a sequence

**-> SPARK STREAMING**

* There are 2 types of data
  + Data at rest : any data that is already stored in a system, usually the analysis we do is on data at rest, batch processing
  + Data in motion : real time processing coming into picture,
    - Use Cases : Online Ads, Credit Card Frau Detection, Online Sales Event
* It is library in spark which helps to do real time analytics
* Other tools : flink, storm(not really popular these days) ,kinesis, kafka streams,
* This is near real time , ie not exactly real time but close to real time
* This process the data in a time interval
* Like : process the data every 5 sec or every 1 sec, but we cant say every message
* They are more into ESP(Event Stream Processing )
* It is a library available like spark sql,
* In spark sql basic data structure was data frame
* However in spark streaming basic data structure is called Dstream , which is just an API and is thus an abstraction of RDD’s
* In spark streaming we need to create an object called streaming context
* Once we have that object we can connect to any source and start processing that data
* It does micro batching I.e if we want to process the data after every 5 sec, then data for the last 5sec is collected and then is processed as batch by spark streaming
* Once we create a streaming context we cant modify it
* Streaming context can not be restarted once stopped
* For one application we use only 1 streaming context
* Stop on streaming context also stops spark context if optional parameter is not set to false
* Transformations which are performed on rdds work similarly with Dstreams
* **Window Based transformation :** instead of just using batch interval we will aggregate the data for a particular period of time
* Window interval : determines how much data we need for the calculation
* Sliding Interval : determines when the final result is calculated
* Spark Streaming is of different types depending upon the source of data
  + **Network** : socketTextStream - to monitor a port number and get data from this port
  + In linux there is a utility called NetCat which is used for reading and writing to network connections using TCP or UDP, thus it allow us to send and receive data from a port
  + Nc -lk 9999 will open the port number this is used to test
  + In production we don’t use netcat
  + **Text File** : spark streaming application will continuously monitor a directory in hadoop and will read the file created in that directory as the file is created
  + Sc.xsetLogLevel(“WARN”) - to display the necessary info only and no warnings
  + Ssc.textFileStream(path)
  + textFileStream is the only application where we don’t have a receiver
  + The moment file appears it will analyze it
  + textFileStreaming is not preferred
  + It does not depend on the type of the file it will process the file
  + **Twitter :** to analyze real time trends and what is going all around the world, sentiment analysis is used to understand how people react to a particular product or a service
  + Stop words dictionary is used to remove neutral words
  + Compare the tweets with the dictionary and remove the neutral words
  + We want to remove max neutral words
  + There is a command “**sbt.run**” it will run the sbt file
  + If our app requires any data we can store that data in resources folder in a sbt application
  + We need extra library dependencies to work with twitter using spark streaming and we specify those dependencies in build.sbt file in an sbt application
  + TwitterUtils.createStream(streamingContext, None) to create a stream from Twitter, in place of None we can pass a filter
  + When we get the data from twitter it is a json object
* Spark streaming has its own ui
* On stopping spark streaming context spark context also stops but to prevent this use ssc.stop(false)
* **Note :** Structured Streaming is the new addition to Spark Streaming - we can create Dstream as Data Frame and we can write SQL queries over it
* **Note :** there is concept of broadcast variables in spark in which we broadcast files which are relatively very small compared to the data streaming to all the worker nodes so that analytical task’s performance increases and the speed of performing the analytical tasks.This is done only when these files are very small in size and data is very huge
* **Note :** in any spark app first we need to set up the configuration of spark(setAppName,setMaster), then the spark context and then spark streaming context

**-> APACHE FLUME**

* It is a data collection utility
* Its an open source project and main purpose is to collect real time data
* Its basic job is to point to point delivery
* Runs on a single machine inside jvm
* One source and one destination
* Also it can not modify the data and send the data efficiently
* It is used to collect log files in real time and send those data (stream into) hadoop or spark
* For its working we need to configure agent
* Agent is configuration file for flume
* It has source channel and sink
* Source is config to get data
* Sink is config to send the data
* Channel is used to buffer the data channel is ram