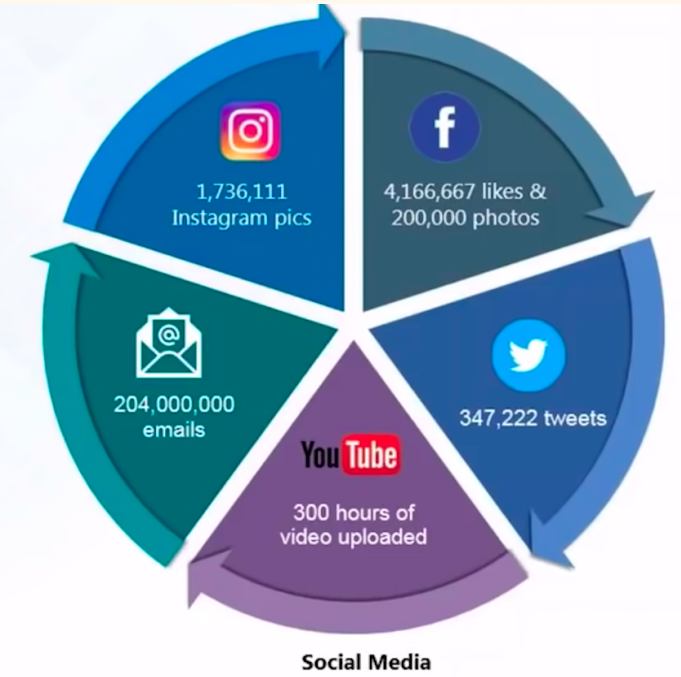
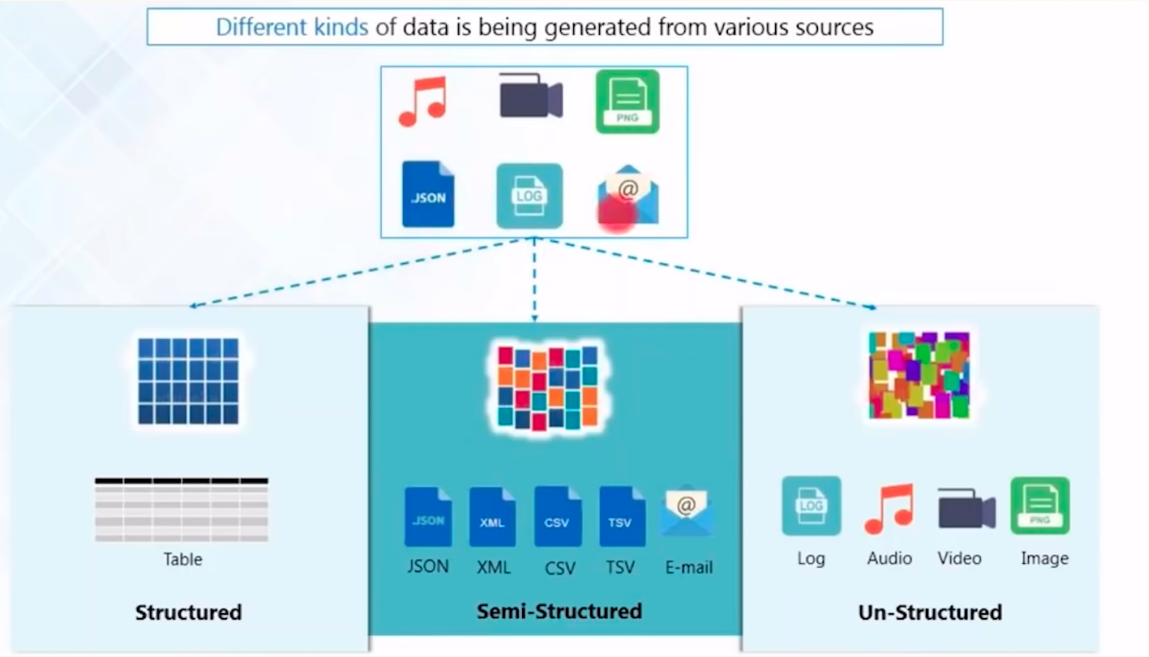
Big Data

What is Big Data?

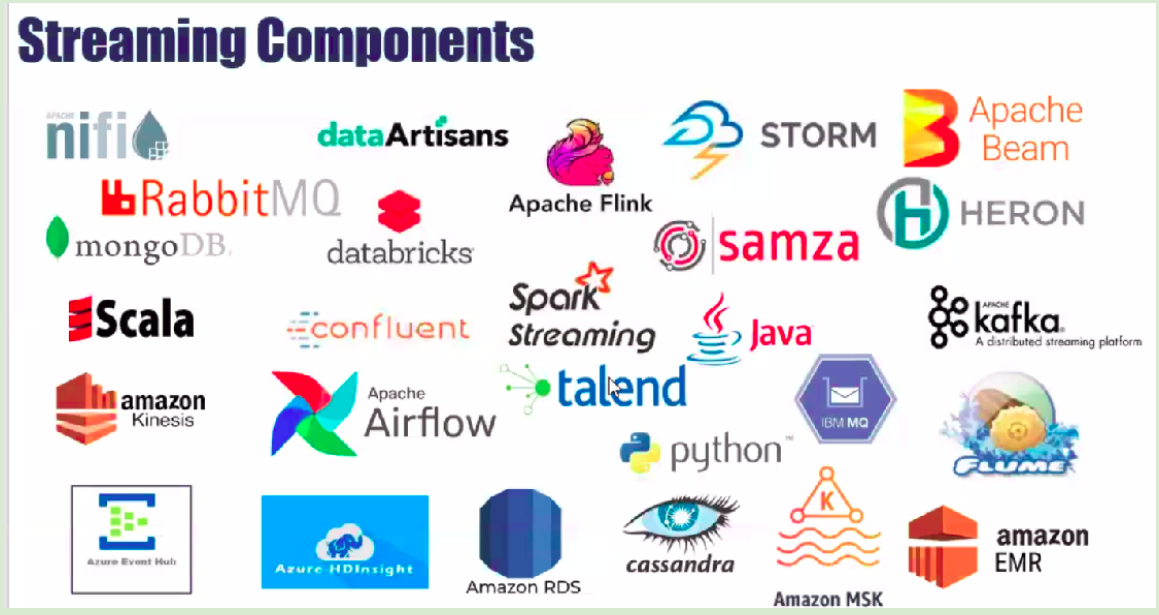




Everyone is creating almost 40 Exabytes data per month

Careers in Big Data -

- Data Analyst
- Data Scientist
- Big Data Architect
- Data Engineer
- Hadoop Admin
- Hadoop Developer
- Spark Developers



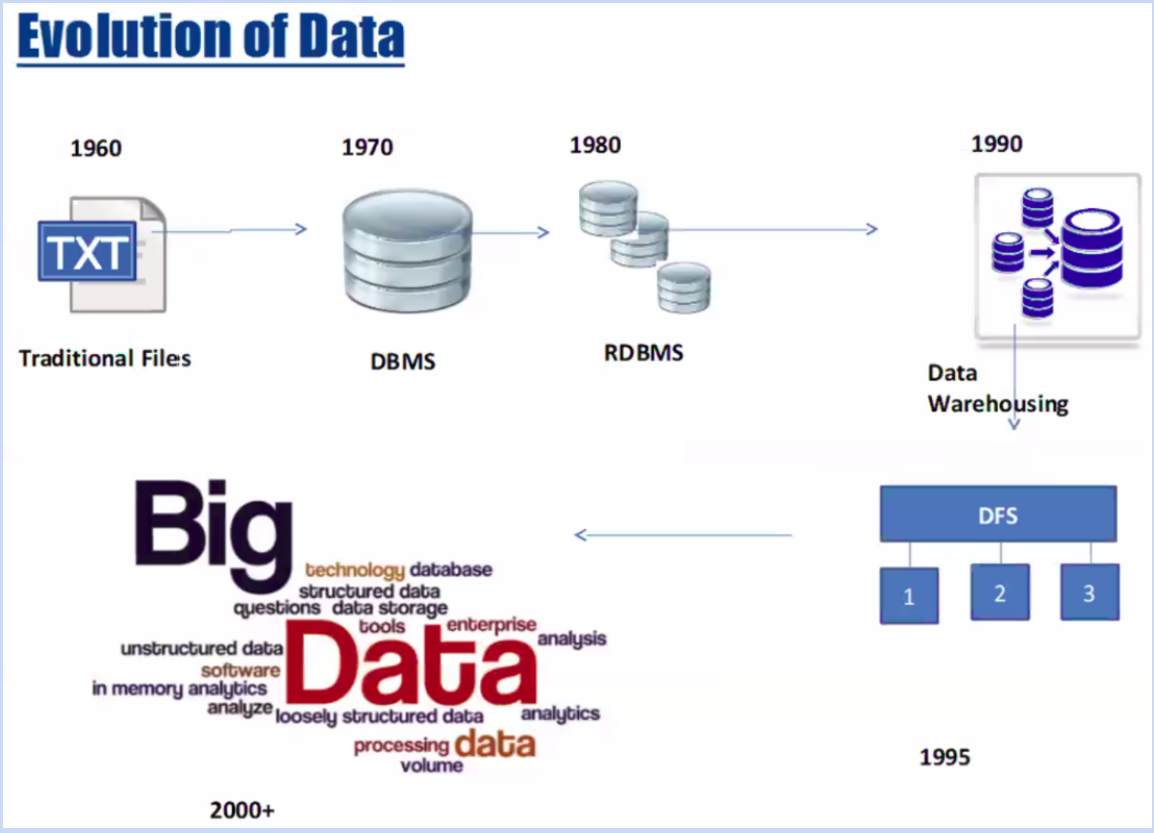
Big data Evaluation -

- 1960 - txt files
- 1970 - spreadsheets/DBMS
- 1980 - RDBMS but Licenced
- 1990 - Data warehouse (many RDBMS) - Damm costly
- 1995 - DFS found by Doug Cutting but failed due to processing wasn't good
- 2003 - Google released GFS paper same as DFS == GFS
- 2004 - Google Released Mapreduce paper → Distributed Processing → Doug Cutting read the paper and was Shocked
- 2005 - DFS enhanced and use Mapreduce
- 2006 - DFS + Mapreduce = Hadoop (name on son's elephant Doll) , HDFS and Mapreduce
- 2008 - Emerged a lot -- Problem -- Hey Mapreduce code is damn hard to understand JAVA —> Doug Cutting said --> do not worry I have a solution to simplify MapReduce

—> Learn SQL —> Trigger SQL —> I will ensure my TOOL converts SQL query to Mapreduce JAVA

- I found a tool, it looks like SQL but not --- If you trigger any queries -- it converts into a Mapreduce JAVA
- In 2008 - Hive and Pig came into the market
    - Mark Zuckerberg --- Facebook using --- HIVE
    - Yahoo--using -- PIG (Not Popular)
- 2009 - **CLOUDERA -** Problem was --for installation and Maintenance -need Laptop
    - Give me Laptop But Installation, maintenance, and Upgrade - are Paid services
- 2011 - **HortonWorks -** Give me laptop and Installation —> Free, Maintenance and Upgrade are Paid
- 2014 - **SPARK** - Birth of a super Power
    - Superpower, Process data very faster, Free of cost, Will perform SQL, Will support Streaming, Will support Machine learning, Can run ON Hadoop
- 2016 - **Cloud** - I will give u **laptops,** I will **install**, **upgrade**, and **Maintenance**
    - AWS EMR - Single Click of Button Hadoop Laptops will be ready within 10 Min by AWS
    - Azure HDInsights
    - Google cloud platform - GCP
    - Alibaba cloud
- 2017 - Cloudera Bought Horton works
- 2022 -- We live in the Cloud Big Data market = Cloud-based Hadoop Big Data analytics

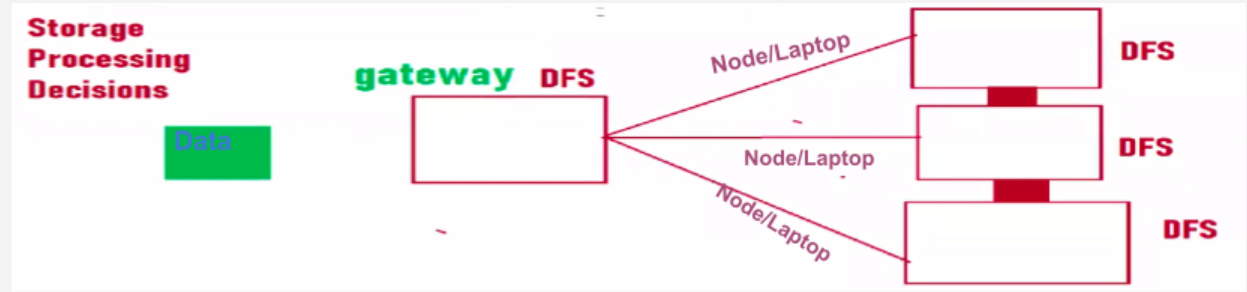
Cloud Hadoop Spark ===> Crazy Technology today



What is DFS -

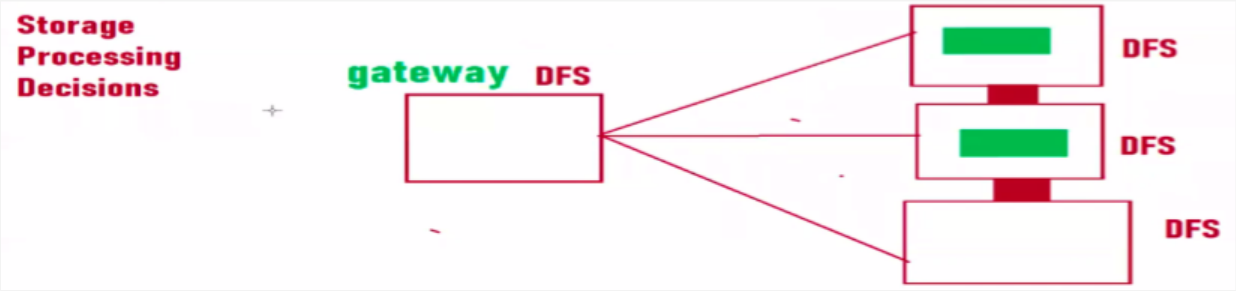
Introduced by **Doug Cutting** in **1995**.

- License-free
- We don't need RDBMS servers
- How DFS works - for example, there are 4 laptops - 1 is Gateway and the other 3 are interconnected to each like the below image -
    - Gateway - Once you give data, data gets stored on the Gateway node first then after running a few commands it is distributed in other 3 nodes.



Step 2) - Put data into Gateway node

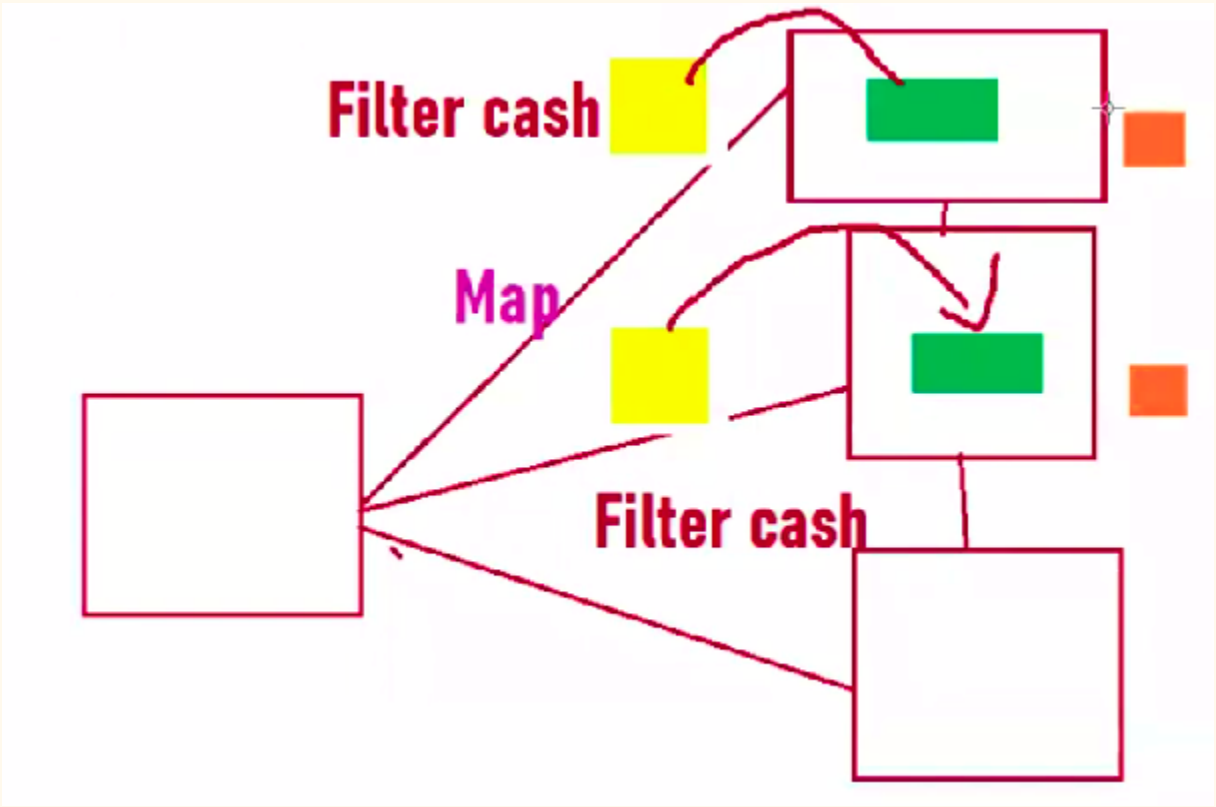
Step 3) - Gateway will divide data into Nodes



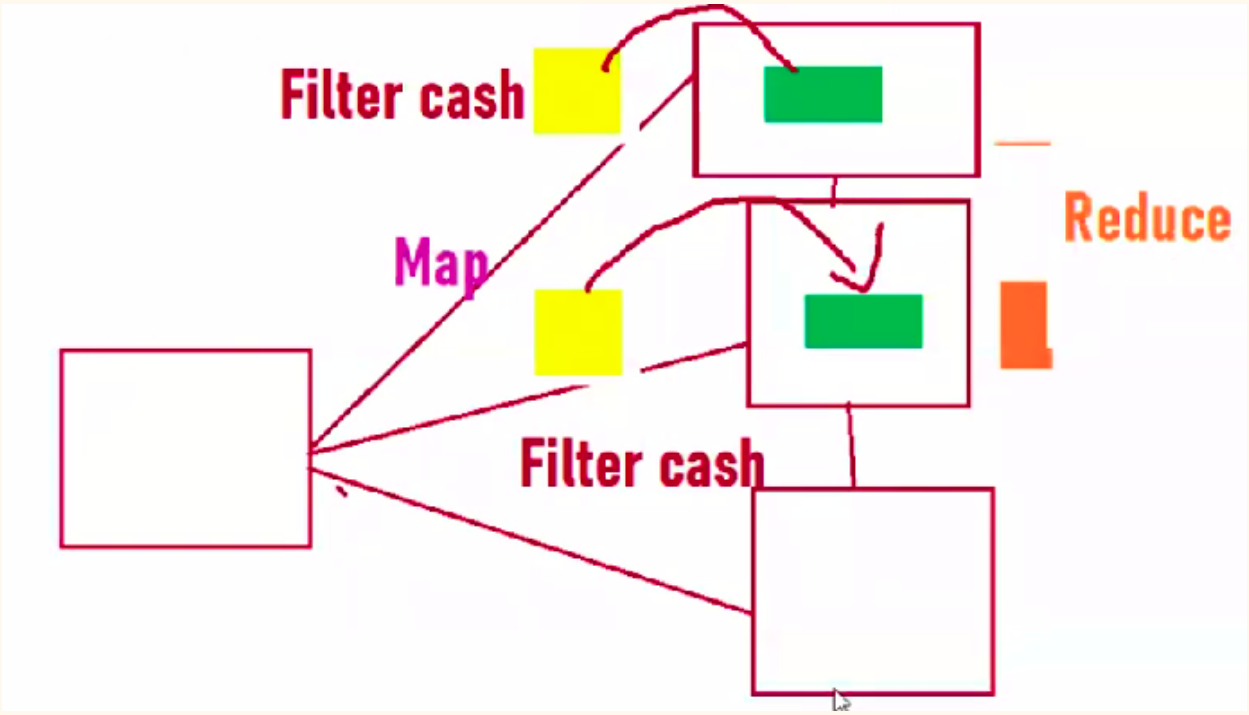
- **Why DFS failed** - As per Doug cutting if data increases then have to increase Gateway Node size as well to process the data. This was really an issue, as putting a lot of money into Gateway Node

How is GFS different from DFS -

GFS(Google file system), they are using the same mechanism to store data as DFS but for processing data GFS uses Map-Reduce. It means data won't come to process instead, the process will go to data and Map it after collecting data from all nodes data gets reduced and returned.



Data is getting reduce -

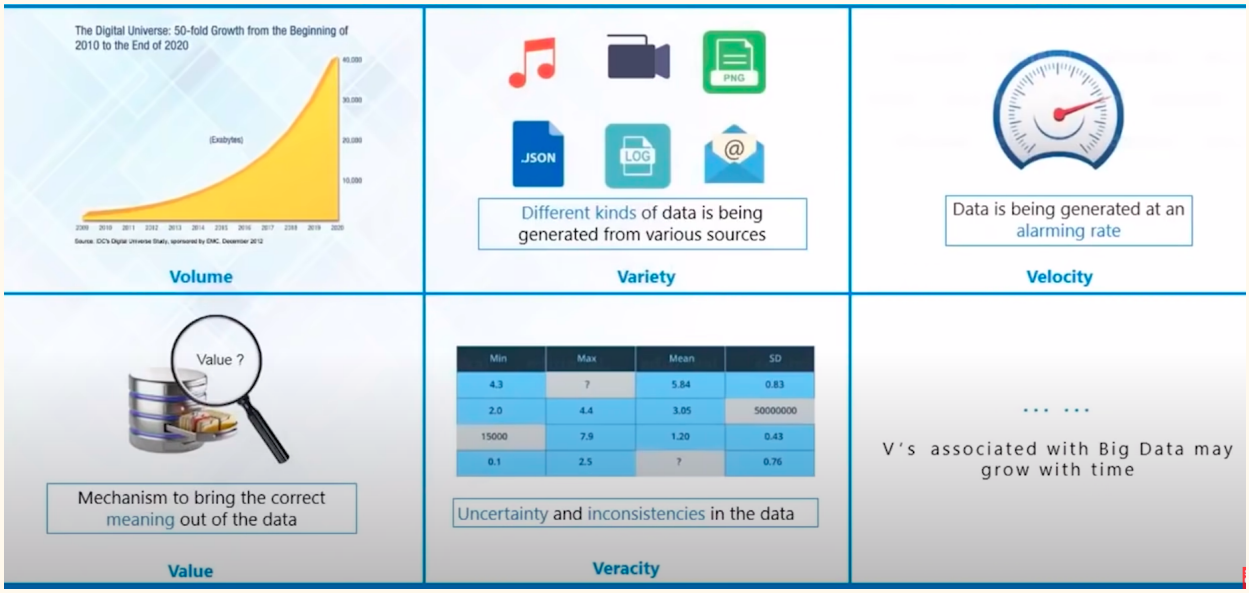


Why Big data?

- ➢ 80% of data is unstructured which is challenging to analyze.
- ➢ Structured formats limitation in handling large quantities.

5 V's - (now it's 6 V's)

1. Volume
2. Variety
3. Value
4. Velocity
5. Veracity - Data can't be perfect in a realistic environment, it will have inconsistencies, errors, and noise. For instance, collecting data for marketing surveys but there can't be data from fake accounts.
6. Variability - Data flow is inconsistent with period peak. The same tweets can have different meanings based on the content.



Challenges of Big data -

1. Cost
2. Handle a variety of Big data - structured and unstructured
3. Scalability
4. Store the sheer size of data

5. Process the huge data - There is no sense in just storing lots of data if we can't process it.

   "That's when Hadoop came to the rescue."

Characteristics of Big data -

1. Hard to store - (Solution) HDFS
2. Hard to Process - (Solution) MapReduce, Apache spark

Big Data layers -

1. Storage
2. Processing/Analysis
3. Testing
4. Data science/ML/AI
5. Automation/Scheduling

Big data Use cases -

Use case 1 - Execution 1

WallMart -- Got a use case
I have problem to Solve
I have two tables in a system ---
cashtable -- lot data for cash customers
creditable -- lot data for credit customers

I have target table (Empty) -- hivetar
Can i get a help to process above two tables and get top 10 customers in the target table
I seek Help

Doug Cutting - Dont worry I have solution
wallmart -- what is it ?

DC ------ Hadoop (HDFS & MR)

WallMart --- Can you show your Hadoop?
Goahead I have question ? How come data will reach your system

DC --- We have super ingestions tool something ---**SQOOP**

WallMart ---- ok you dump the data .. who will process in system ?

DC -- Mapreduce to file first 10 customers --- I am going to use tool know as **HIVE** which uses Mapreduce

- Ingestion Tool - **SQOOP**
- Process Tool - **HIVE**

WallMart --- Go ahead ---

DC ---- Hey wallmart -- I have used sqoop to bring data - I have used hive to process and top 10 customers I have found

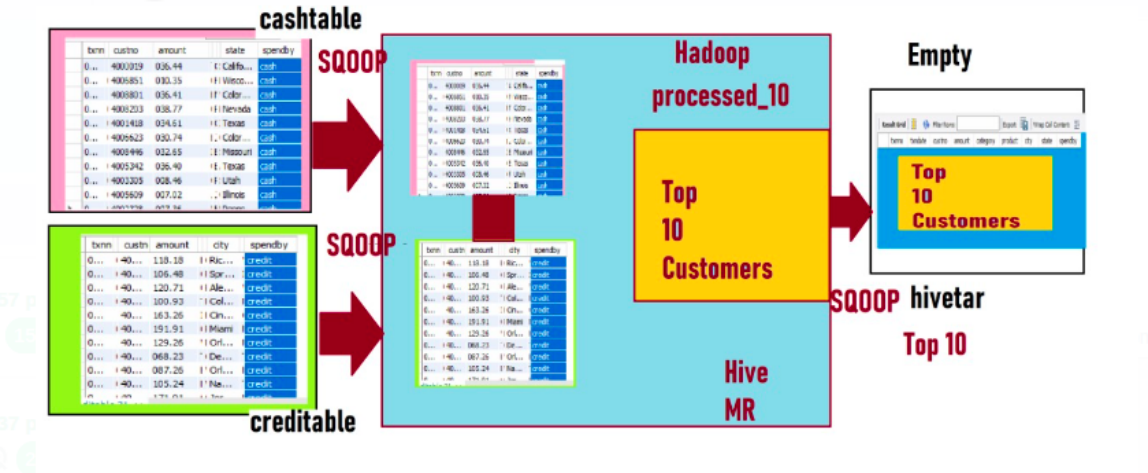
WALLMART --- Where are those top 10 customers -- Is it in you hadoop system

DC --- Yes IT IS Sitting in our hadoop system-- Dont worry
I will use the same **sqoop Tool** to put the top 10 customer to your target system
Its Done

WallMart ----- KUDOS

WALLMART -- Doug Cutting you did well. I will give business To you

DC used followed below method -



Hey This is Jeff Bezos. (AMAZON)

I have another powerfull system —
My system complets the work very faster. Same scenario I can assure I will use damn powerful tools

WALLMART --- What are your Tools

Jeff Bezos --- I am also going to use Hadoop Only. But I going to use it differently

WALLMART --- Tell me your Design like DOUG CUTTING

Jeff -- Give me target Table

Wallmart -- created --- sparktar

Jeff -- I will use my own system S3 AWS

WallMart what is the ingestion you gonna use what is the processing you gonna use "?

Jeff --- Both are powerfull to me
Ingestion ----- NIFI
Processing ---- SUPER POWER --- Spark (Hadoop)

Wallmart --- can you show your system

Jeff --- Sure

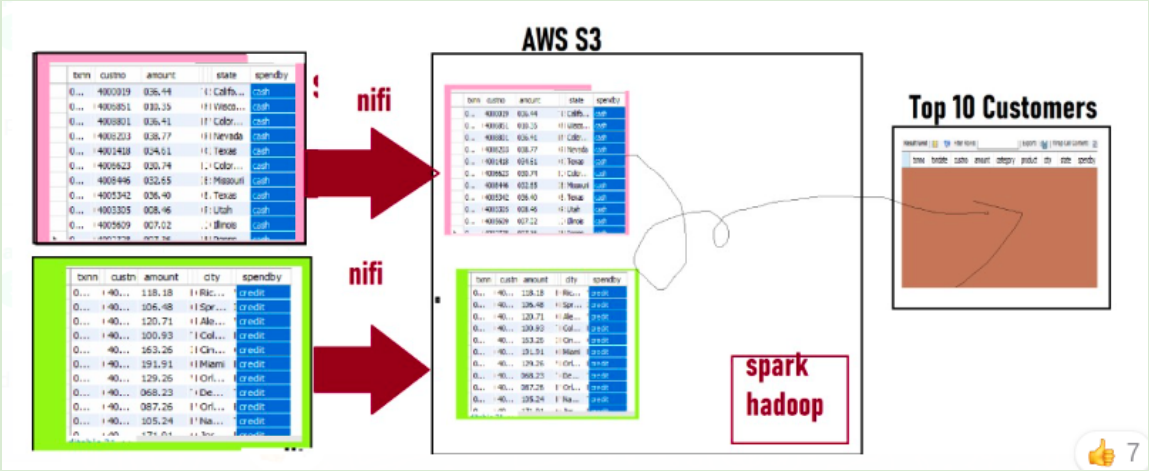
Wallmart --- Go ahead ---

Jeff -- within 3-4 Min-- He said its Completed

WallMart --- Where is the data -- is it in your system

Jeff --- My super directly written your target system

Wallmart --- Perfect I am giving the contract to you



Software needs to install for Big Data learning -

- **Putty/Mobaxterm**

- VirtualBox → install Cloudera inside VirtualBox
- **Windows** -
    - 7zip

Programming topics need to learn for Big data -

- [x] ~~Architecture~~
- [x] ~~Access modifier~~
- [x] ~~OOPs~~
- [x] ~~Exceptional Handling~~
- [ ] String Handling
- [ ] File Handling
- [ ] Collections
- [ ] Functional Programming
- [ ] JDBC connectivity

Vim editor (Edit a file)-

- **Vi <fileName>**
- Enter '**i**' to enter in "INSERT" mode
- To exit from the file press **ESC**
- then enter "**:wq!**" to exit from vi editor

Hadoop

Stored -- Processed- Decision Making

Hadoop is a framework that uses distributed storage and parallel processing to store and manage big data. It is the software most used by data analysts to handle big data, and its market size continues to grow. There are three **components** of Hadoop:

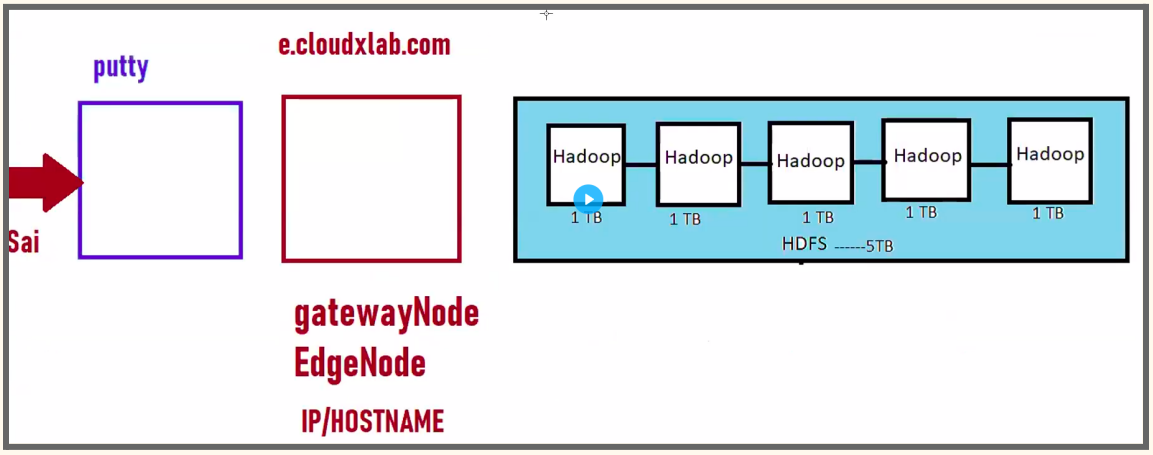
1. Hadoop HDFS - Hadoop Distributed File System (HDFS) is the storage unit.
2. Hadoop MapReduce - Hadoop MapReduce is the processing unit.

3. Hadoop YARN - Yet Another Resource Negotiator (YARN) is a resource management unit.

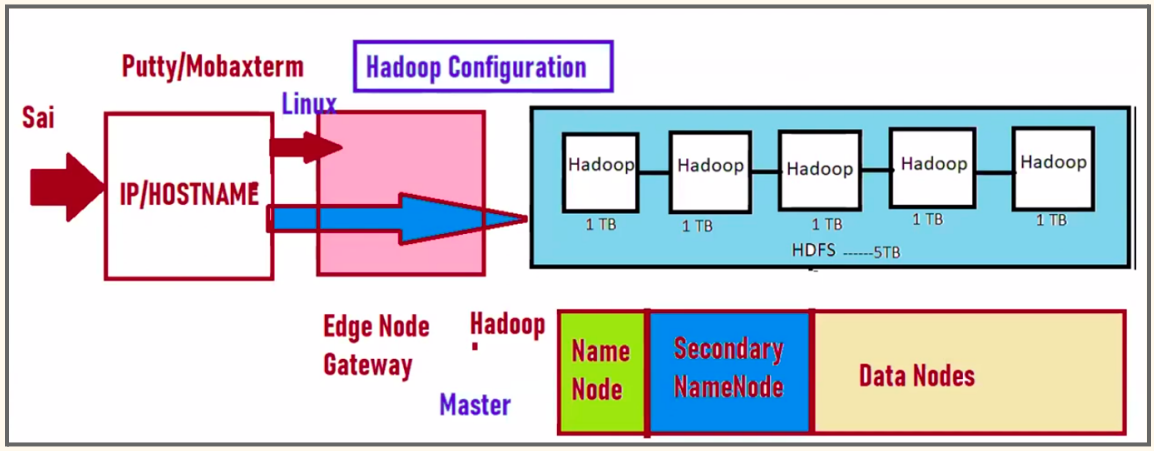
Cluster - Group of Nodes/laptops (including gateway node)

To login into Gateway -

1. Tool to use for login into Gateway - **Putty**, **Mobaxterm (**Linux/Mac users can also use **ssh@hostname** in the terminal**)**
2. you should have a hostname or IP address and username/password.
3. If want to talk to **Gateway** then use - **Linux** command
4. If want to talk to **HDFS/ Hadoop-Node**s then use - **Hadoop** Language
5. **For example, we have 6 node cluster in this 1 node will be the gateway node/Edge node to communicate to the other nodes**, like image (1)



In other 5 nodes, 1 node will be Name node, 1 will be the secondary node and remain 3 will be data node, like below image -



Mail - Use Case

Manager --- Sai,, Cluster is ready --- 6 Node Cluster .. Can you see whether you have any data in the GateWay Node
Sai --- Sure Boss - I see whether we have any data in the Gateway Laptop. But can you help how to check it.
Manager ---- Sai,every laptop has a ip address or HOSTNAME . Use the gateway hostname-- e.cloudxlab.com
Sai ---- Thanks for the hostname But how can I login to that gateway
Manager ---- You have a specific Tools to login to the remote gateway node ---- **putty or Mobaxterm**
Sai ---- Dear Manager. I have installed Putty I have given hostname also but it is asking login as →username/password
Manager -- Sai use below username/passwrod –> twezeyo29867 / BG227QNZ
Sai --- Manager. I got Logged in
Manager --- Its very basic that you have log in - Do the work whether you have any data sitting in the gateway node
Sai --- Can you help me how to check the data
Manager ---- Sai, To communicate with Gateway node -- You should know the communication Language
Sai ---- Can I communicate in english

Manager --- GATEWAY nodes will not understand English... GATEWAY LAPTOPS ARE INSTALLED WITH LINUX OPERATING SYSTEM --- TO COMMUNICATE WITH gateway you should have LINUX LANGUAGE
Using linux can you list and check whether you have data

Sai --- I listed the command using ls I got a file name 200MB.zip
Manager -- can you tell me what is size of that file
Sai --- Dear manager - file name is 200mb.zip file size also 200MB
Manager-- Thanks. Can you also check whether hdfs is also having the data ?
Sai ----- Then Shall I login to any of the HDFS laptop and check?
Manager --- Idiot , You do not have access to the HDFS laptop even I will not have if You have to communicate with the HDFS .. The only path is you have go through EDGE NODE
Sai ---- ok.. Shall I communicate in the linux language

Manager ---- If you talk in Linux.. Edge Node will consider that you are talk to Edge Node
. If you want to talk to HDFS -- You have to talk in Hadoop Language
If you start talking in hadoop language with Edge Node
Then edge Node will understand that commands are not for edge node It will automatically sends those commands inside the cluster

hadoop fs -ls

Sai -- yes boss.. I spoke in hadoop language -- I got the output --- hadoopdir
Manager -- Thanks

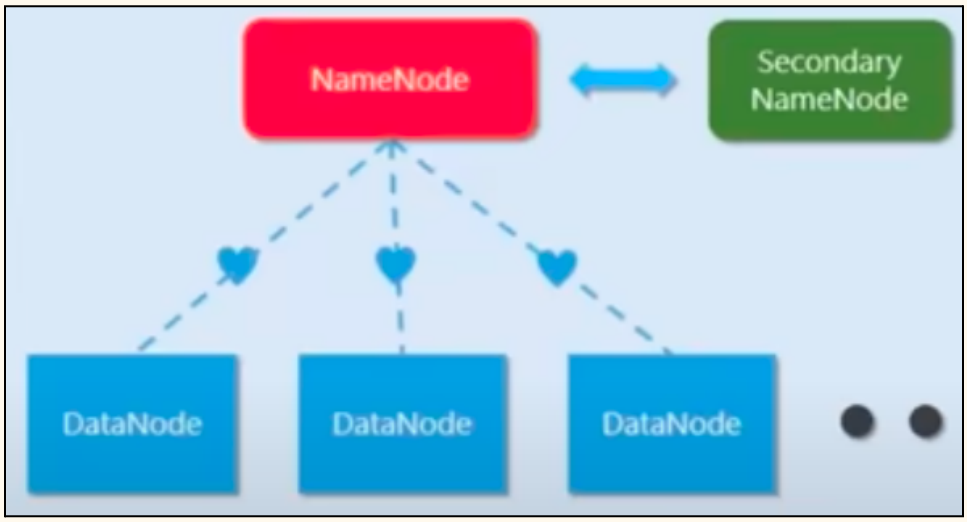
Components of Hadoop

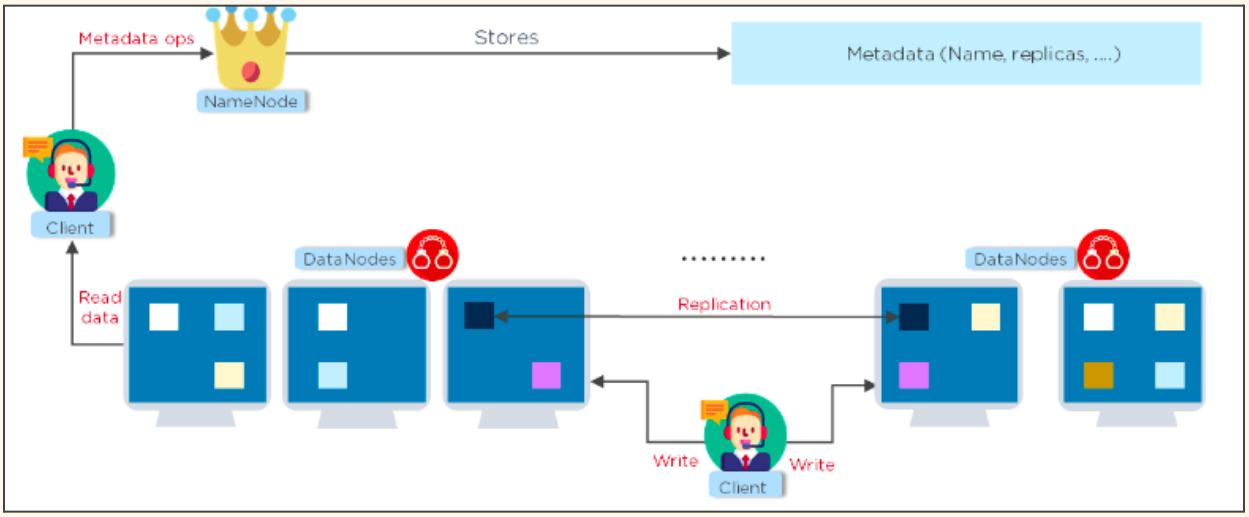
1. **Name node** - Central file system, contains metadata of Data nodes like(which datanode is alive and which datanode has data, etc.)

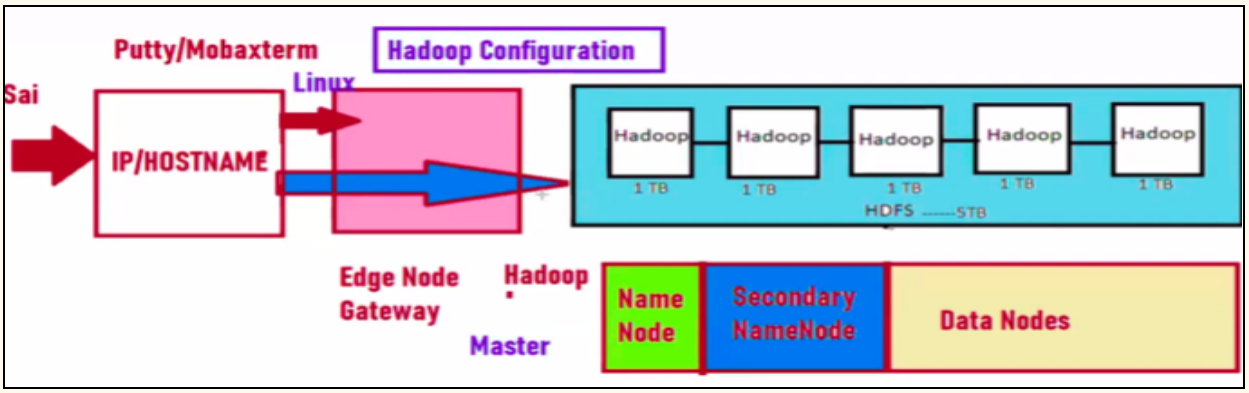
   Rules of Namenode -

   a. Send a heartbeat in every 3 sec to namenode.

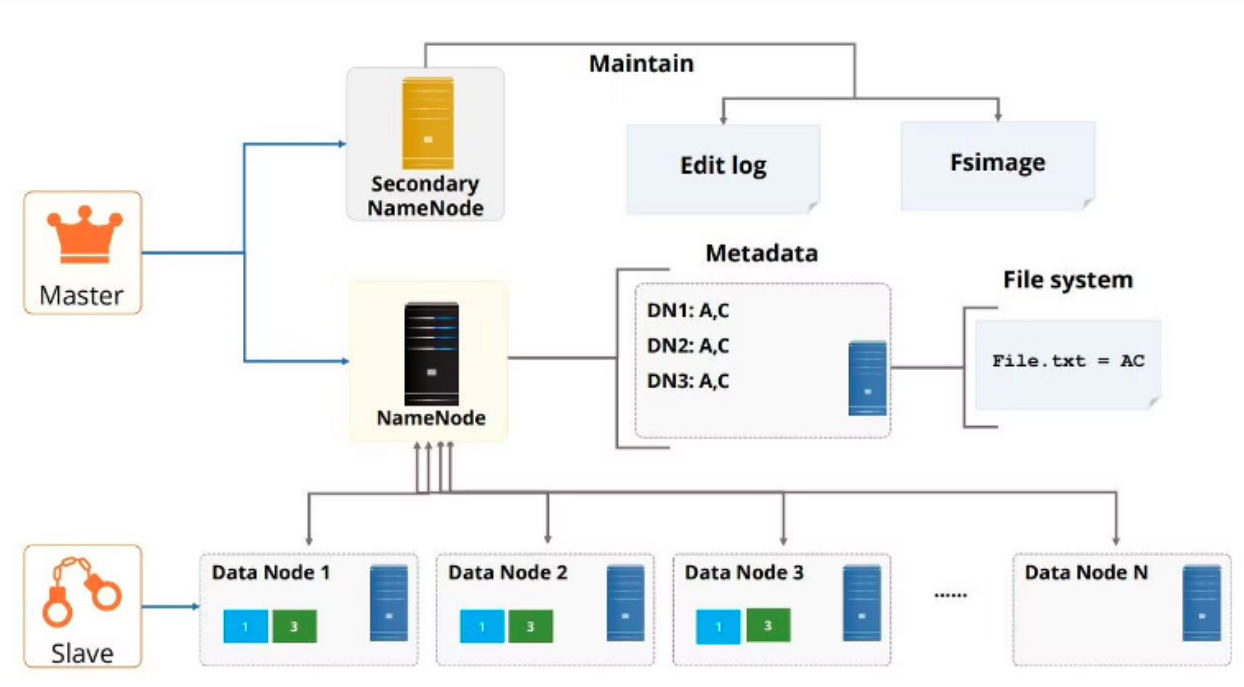
    b. If you don't send a heartbeat for 10 min I will give grace time and still you still don’t send, I will declare you a dead HDFS node.
    c. If you have any work send a copy of the work to 2 other nodes. (There will be 3 copies of Each data)
2. **Secondary Name node** - Data backup of Name Node
3. **Job Tracker** - Centralized job scheduler
4. **Data Nodes** - cluster/machine where data get stored and processed, it is also called **slave** nodes. Every slave node keeps sending a heartbeat signal to the **Name node** every 3 seconds to state that it's alive.
5. **Task Trackers** - a software service that monitors the state of the job tracker



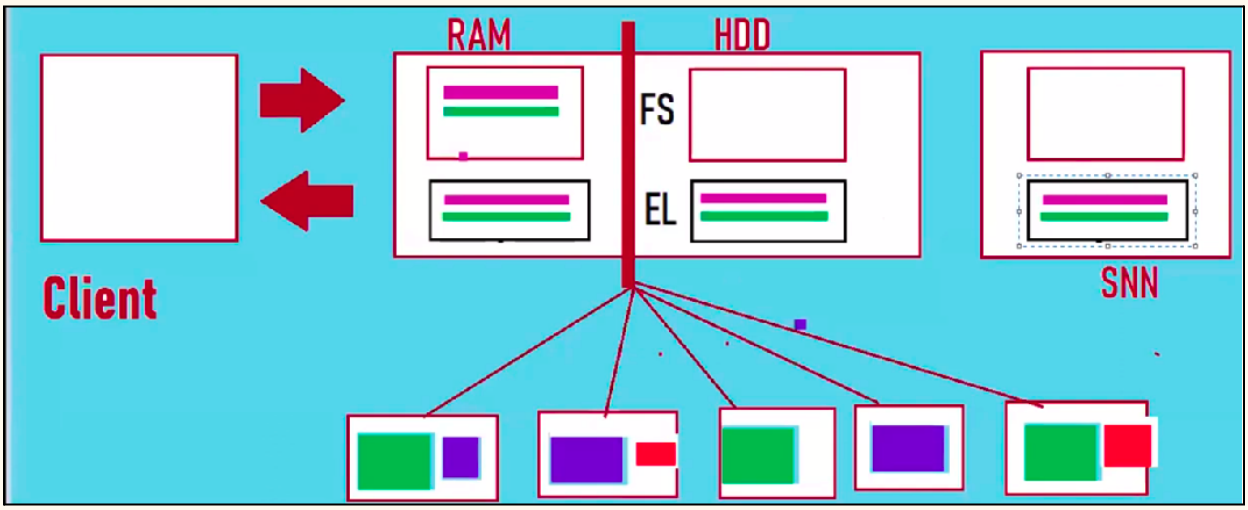




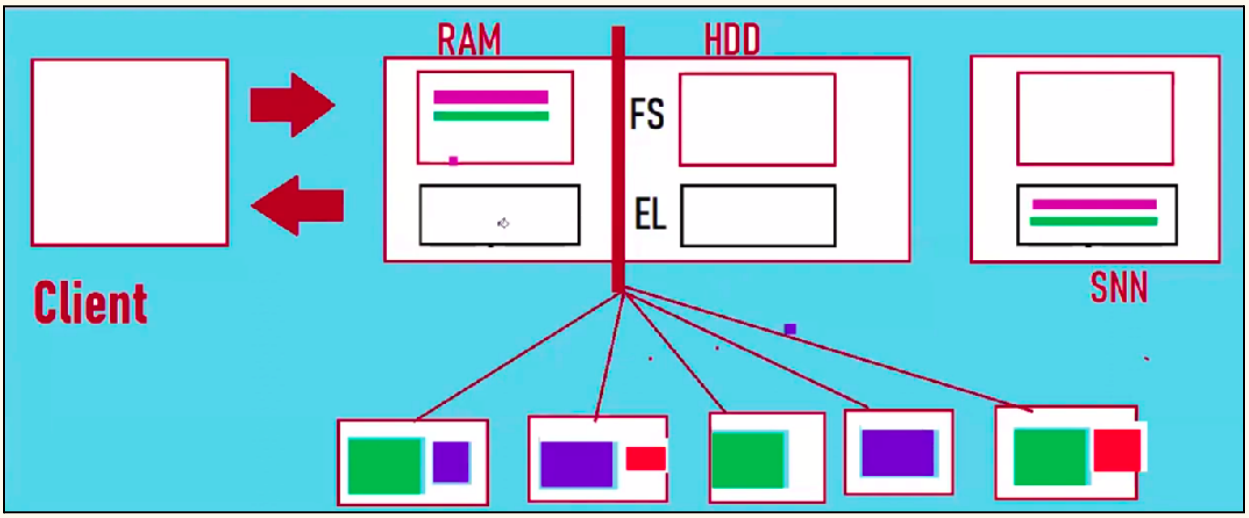
Hadoop 1.0 Architecture -



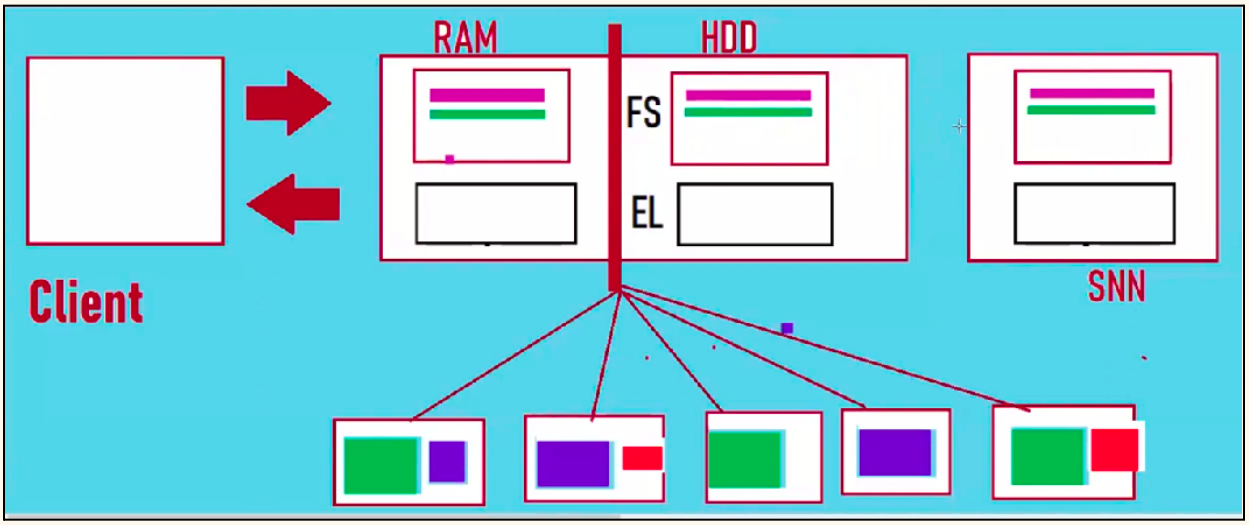
1. Once data translation happens, Save metadata in three places of Namenode -
    a. FS image - RAM
    b. Edit Logs(EL) - RAM
    c. FL - HDD (hard disk)
2. Then After 1 hour,
    a. if this is the first transaction then Namenode sends the FS Image-HDD –> SNN(secondary Name node)-FS image
    b. Latest EL (RAM) —> SNN-EL



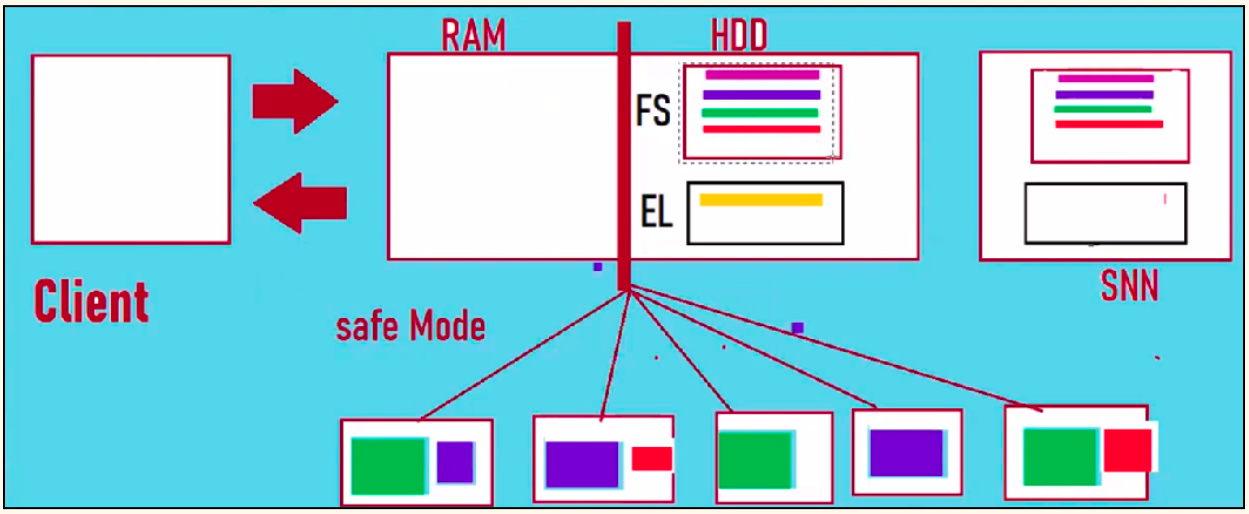
3. And Clear EL - RAM and EL- HDD of Namenode



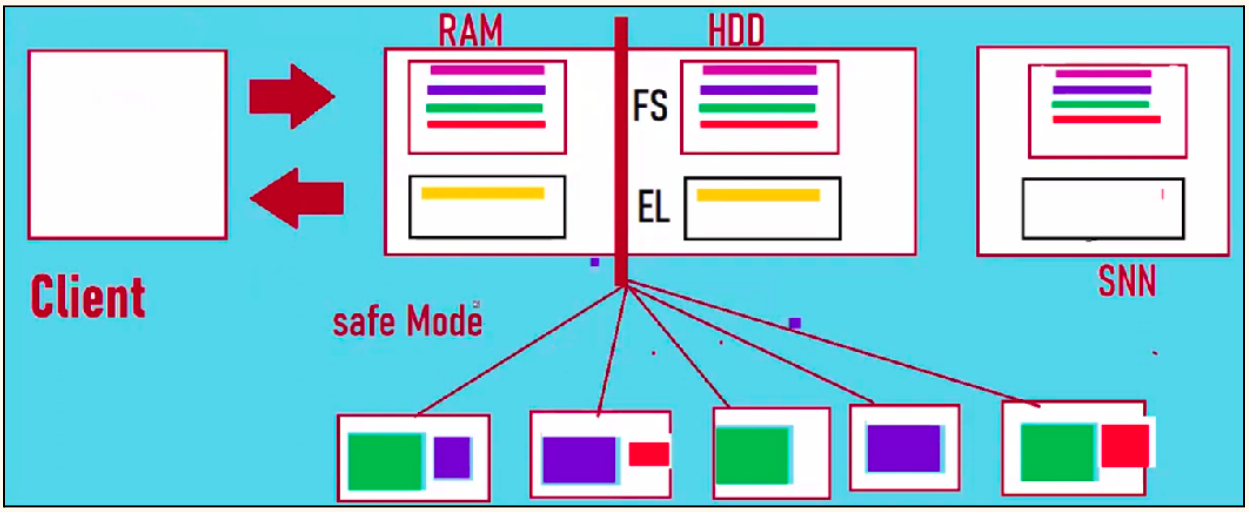
4. Then **Checkpointing(Merge) SNN- EL data to SNN-FS image** and then **send SNN-FS image to HDD-FS image**.



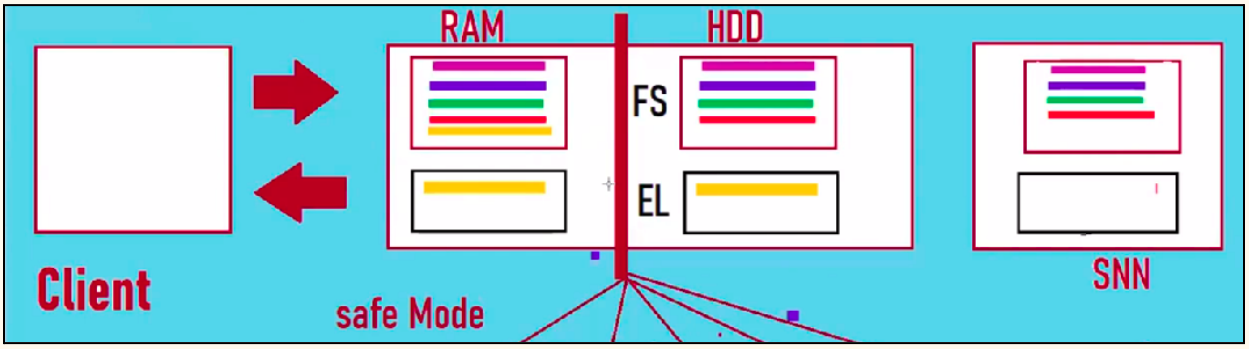
5. What if Namenode- RAM crash, then Namenode immediately goes to **SAFE MODE,**



6. Then Namenode will take FS images and EL from HDD(Namenode).



7. And will Checkpointing(Merge) Data of EL- RAM (namenode) to FS image-RAM(namenode)

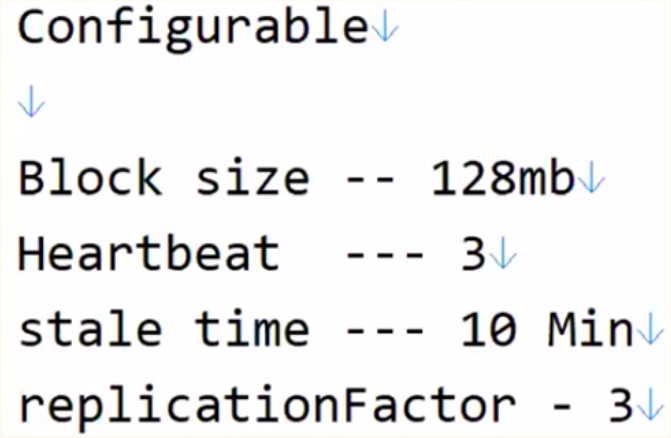


FS image -

- Why FS image can't be committed to the hard disk every time there is new data - because FS image is huge in size and it will slow down the system, also if that time Client asks for metadata from Name node it will not respond properly.
- FS image in Hard Disk should be copied during restart time.

Important Notes for Hadoop 1.0 -

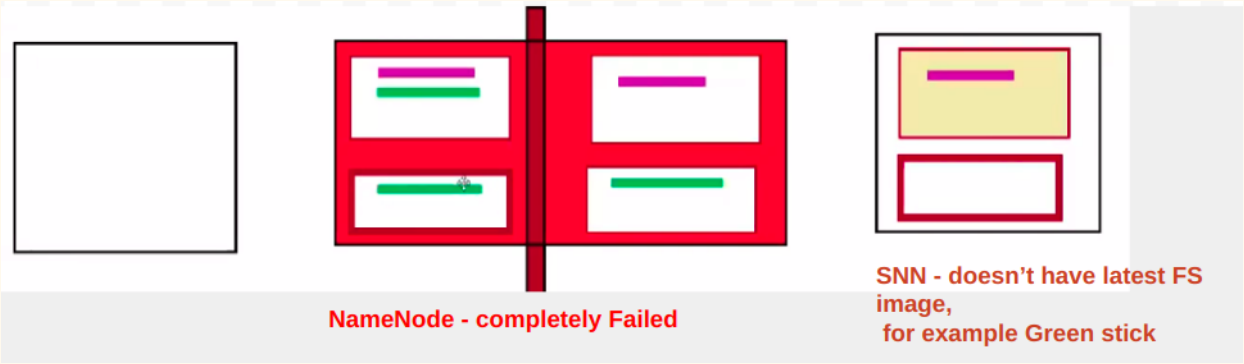
- **Each transaction will be divided into Blocks, and its size is 128 MB(in Hadoop 1.0 Block size - is 64 MB). Suppose the transaction is 200 MB then it will divide into 2 blocks → 128 MB and 72 MB**
- These configurations can be edited and set as per requirement for Edge Node -



- If the transaction is not replicating data to other data nodes then the name node will throw notification failure.

Defects of Hadoop 1.0

- Data Loss - Namenode failure was a threat. Suppose after the New data transaction namenode completely fails, then there is no data recovery in that case as we can't depend on SNN because it doesn't have the latest FS image.



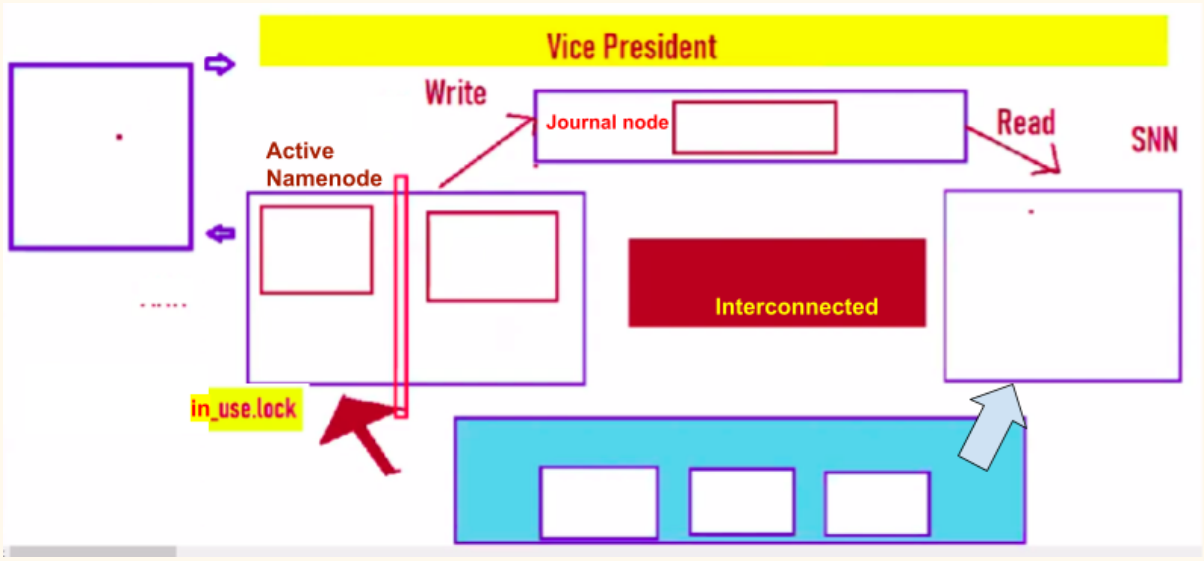
Hadoop 2.0 Architecture

2 New nodes were added -

1. Journal Node
2. Zookeeper

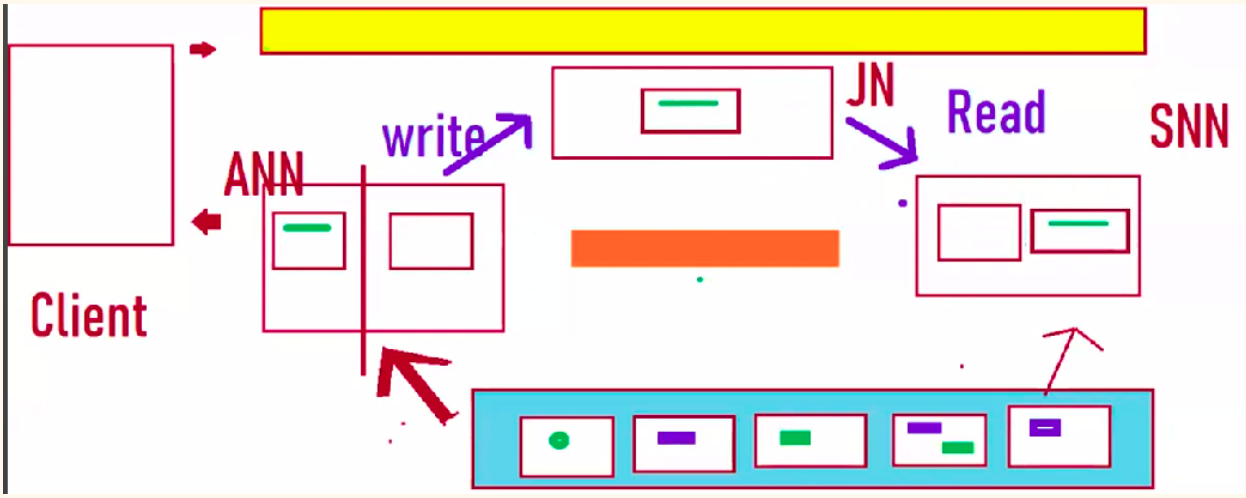
Rules of Zookeeper-

1. Secondary namenode renamed as **Standby node.**
2. Name node(Active name ode) will write Edit logs to Journal Node.
3. Standby name node, During **checkpoint** have to read form **Journal node**.
4. If required(in case of name node failure), the **Standby namenode** has to become **Active name node**.
5. **Datanodes** have to send a heartbeat to both **Namenode** and **Standby Namenode**. But **metadata** will only be sent to Namenode.
6. Whichever is the Active node - it will have file - **in_use.lock** as a representation that this is the current **Active name Node**.
7. ANN and Standby nodes are interconnected.
8. After 1 hour -
    i. Read **EL - Journal node** and write in **EL-Standby.**
    ii. If it is the first transaction then copy the **FS-image** of **HDD-ANN** and save it in the **Standby** node.
    iii. And also clear **EL-Journal node**
    iv. checkpoint **EL-Standby node** –> **HDD-ANN**(active node).

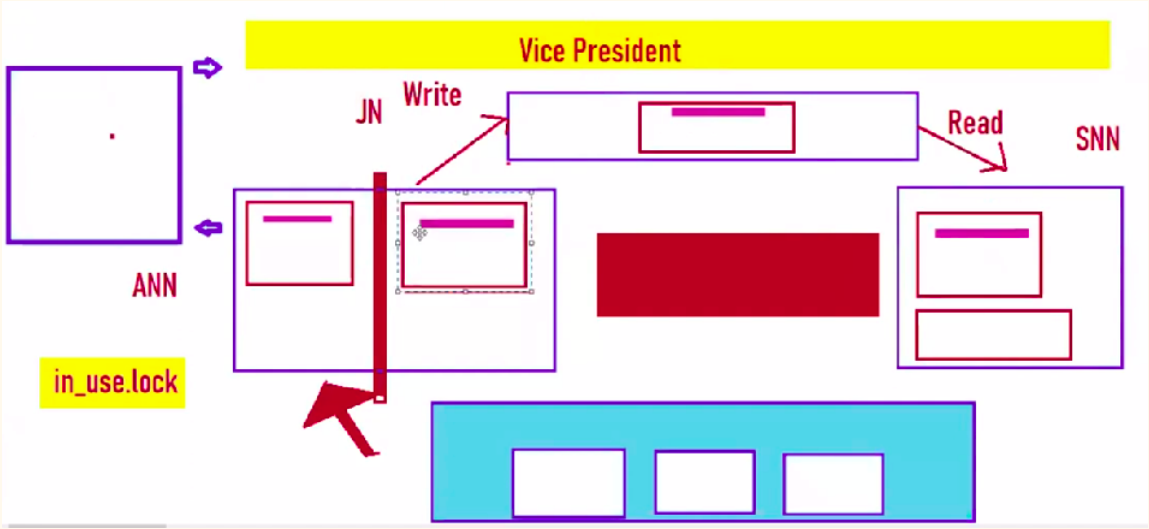


Flow

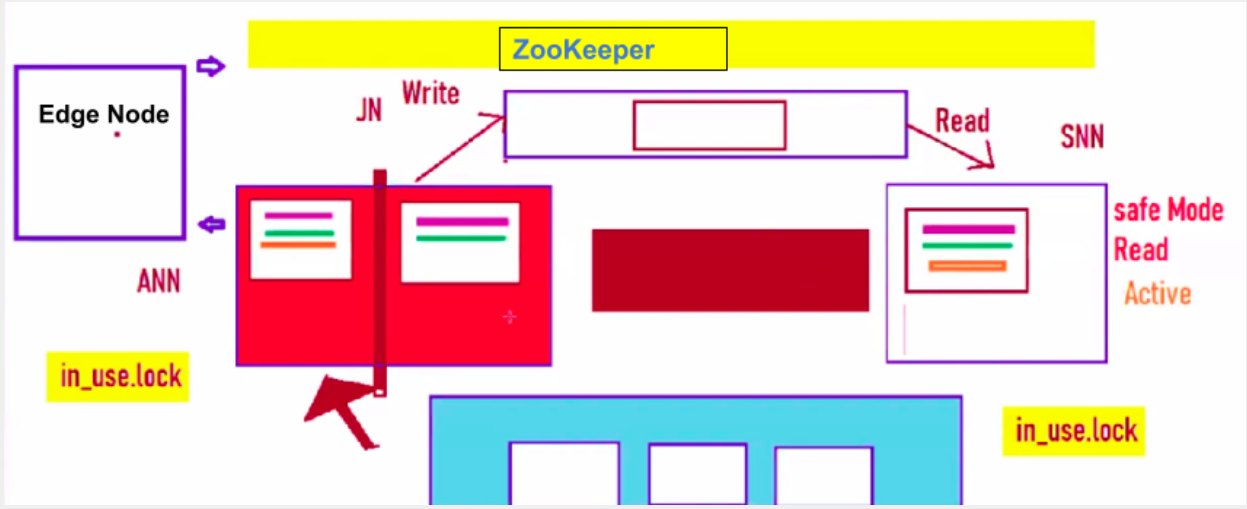
1. First Transaction - Write **FS image** in ANN(RAM)and **EL in journal node** and after 1 hour **SNN** will **read EL from journal node** and copy **old HDD form ANN**(if this is the first transaction).



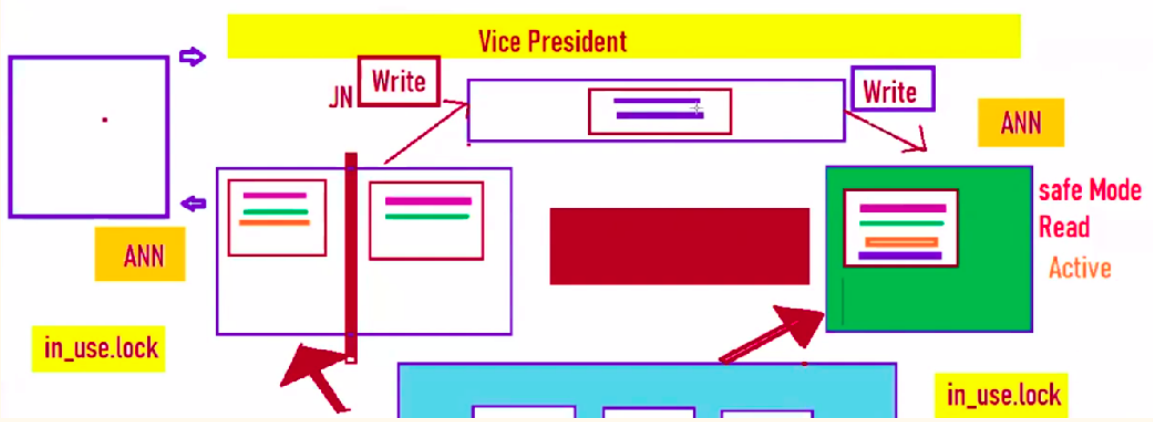
2. After one hour there will be checkpointing where SNN merge EL to FS-SNN and copy **FS-SNN** to **FS-ANN(RAM)**



3. If ANN goes to complete failure - Then the Standby node will go to **SAFE MODE** and a new file wil be created for **Standby node** - **in_use.lock** (representing this is the current active node).



4. What if after some time or hours Dead Node(Name node) becomes active - In that case, Both the Dead(name node) and current Active node(standby) will start to write to the **Journal node,** which means now there will be 2 times EL in journal node, which is an issue. This scenario is called **Split Brain Scenario.**



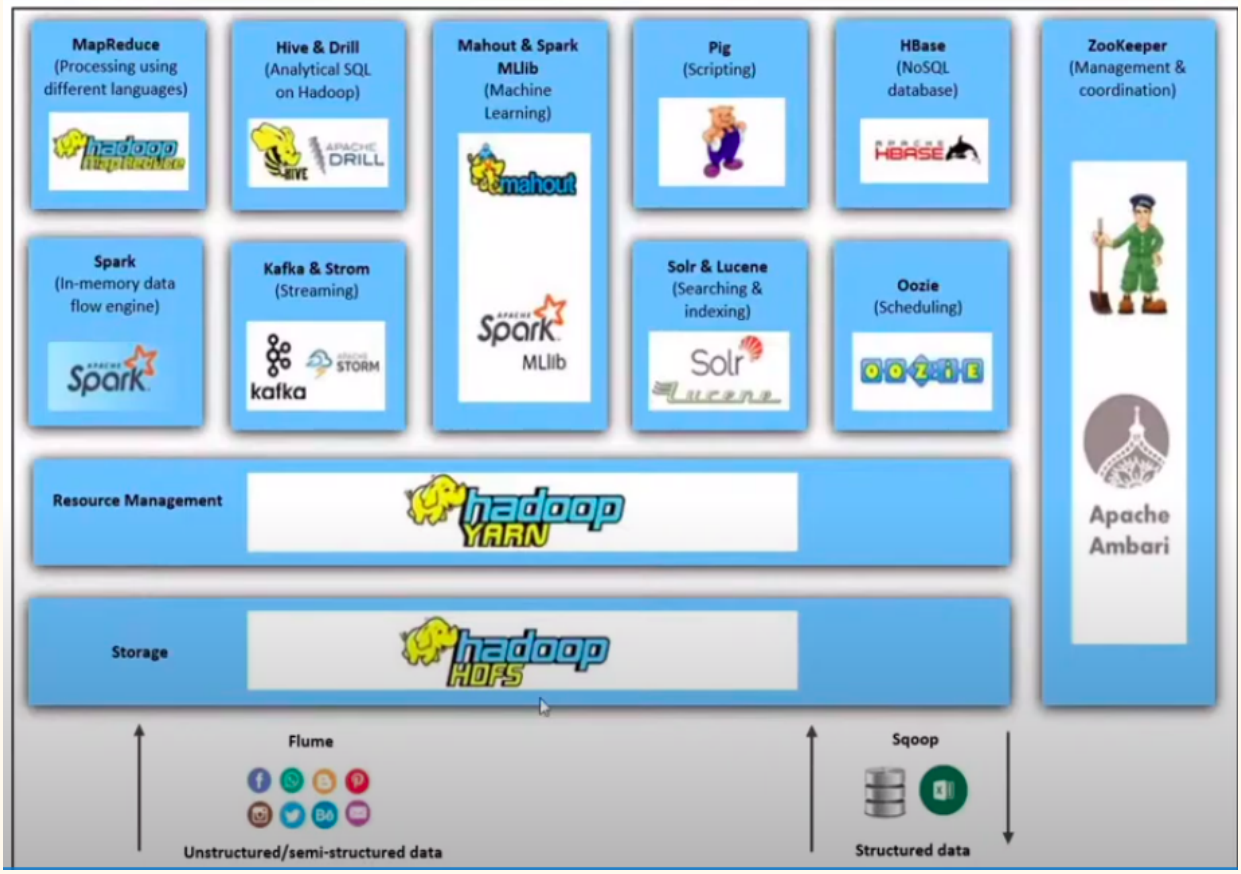
Solution - Doug cutting created a Program **Fencing Java Program** where **Zookeeper** sends messages to the Dead node, where dead mode will stop writing to other nodes. And **the Dead node** is now **Standby node**.

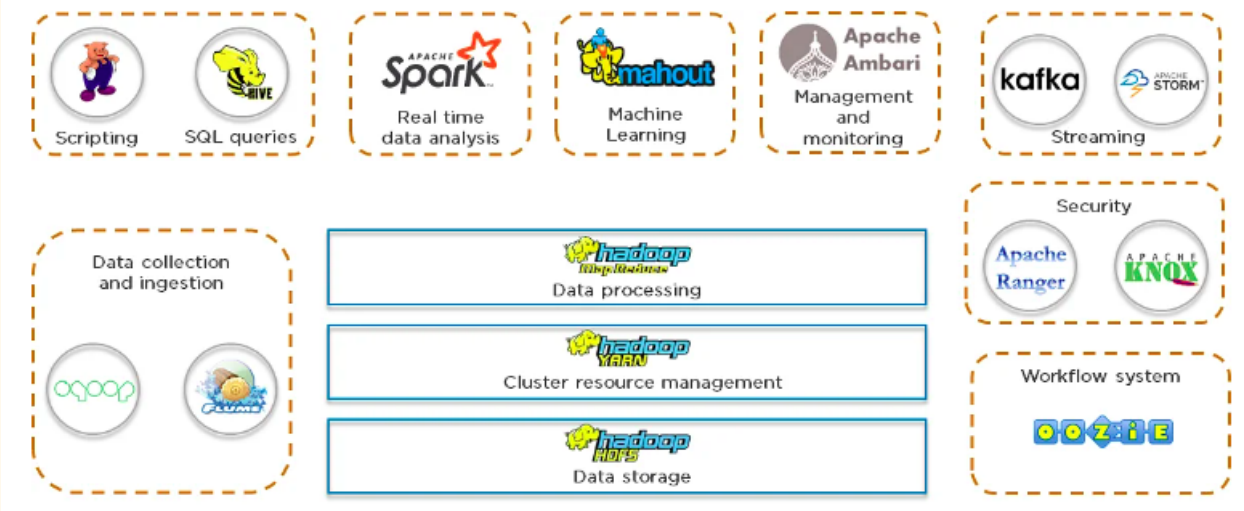
Hadoop Commands (just like Linux)-

To talk to Hadoop →prefix Linux command with '**hadoop fs -**'

- hadoop fs -ls
- hadoop fs -touchz <fileName> // create a new empty file
- hadoop fs -cat <fileName> //print file content on console
- hadoop fs -put <edgeNodePath> <HDFSPath> // copy file from edge to HDFS
- hadoop fs -get <HDFSPath> <edgeNodePath> // copy file from HDFS to Edge
- hadoop fs -moveFromLocal <edgeNodePath> <HDFSPath>
- hadoop fs -moveToLocal <HDFSPath> <edgeNodePath>
- hadoop fs -appendToFile <sourceFilePath> <targetHDFSPath>
- hadoop fs -rmdir <path> //delete folder
- hadoop fs -rm -r <path> //delete folder and file inside it
- hadoop fs -rm <path> // delete file
- hdfs dfsadmin --safemode get //→find out if you're in safe mode
- hadoop dfsadmin -safemode leave //→turn off safe mode
- hdfs dfsadmin -report //→find out how much disk space is used, free, under--replicated, etc.

Hadoop Ecosystem


MapReduce
(Processing using different languages)
Hive & Drill
(Analytical SQL on Hadoop)
Mahout & Spark MLlib
(Machine Learning)
Pig
(Scripting)
HBase
(NoSQL database)
ZooKeeper
(Management & coordination)
Spark
(In-memory data flow engine)
Kafka & Strom
(Streaming)
Solr & Lucene
(Searching & indexing)
Oozie
(Scheduling)
Apache Ambari
Resource Management
Storage
Flume
Unstructured/semi-structured data
Sqoop
Structured data


Scripting
SQL queries
Real time data analysis
Machine Learning
Management and monitoring
Streaming
Security
Data collection and ingestion
Data processing
Cluster resource management
Data storage
Workflow system

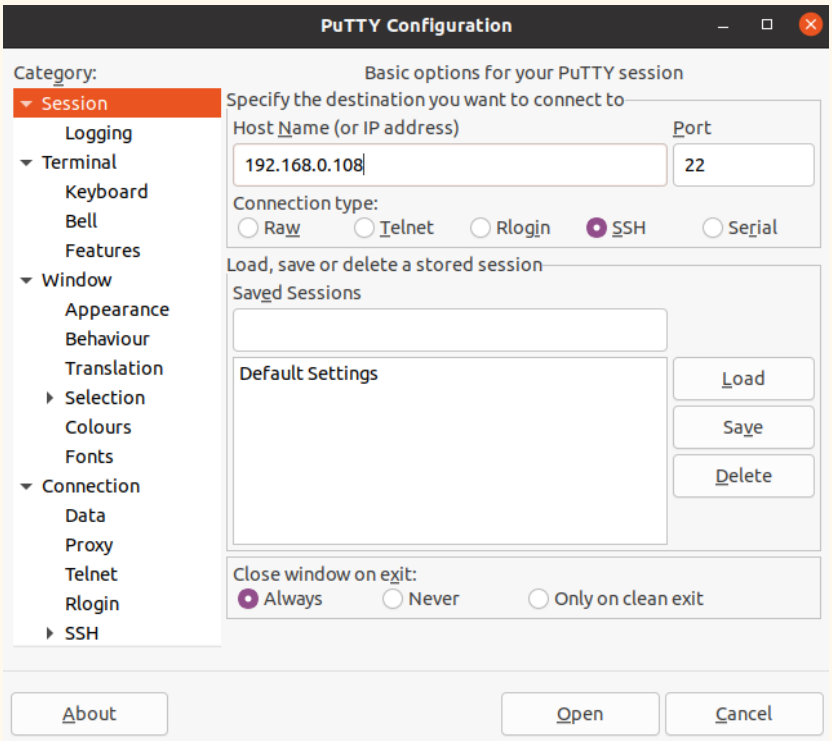
Cloudera

Installation →

- **Prerequisites** - Install virtual box [sudo apt install virtualbox]
- **Article** - https://www.simplilearn.com/tutorials/big-data-tutorial/cloudera-quickstart-vm

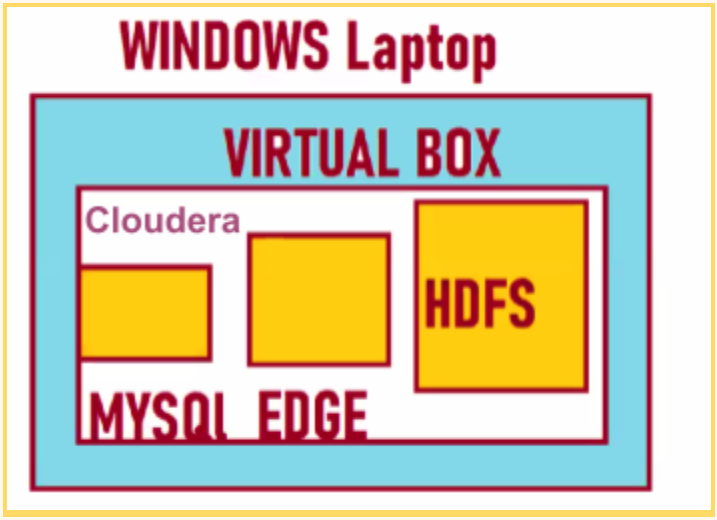
How to Login into Edge node/Gateway node

1. Login direct into Cloudera by opening **VirtualBox** with Cloudera installation
2. By **putty**, **mobaxterm**, or **ssh** as per OS -
    - **Windows** - Open putty and provide **host IP** and connect



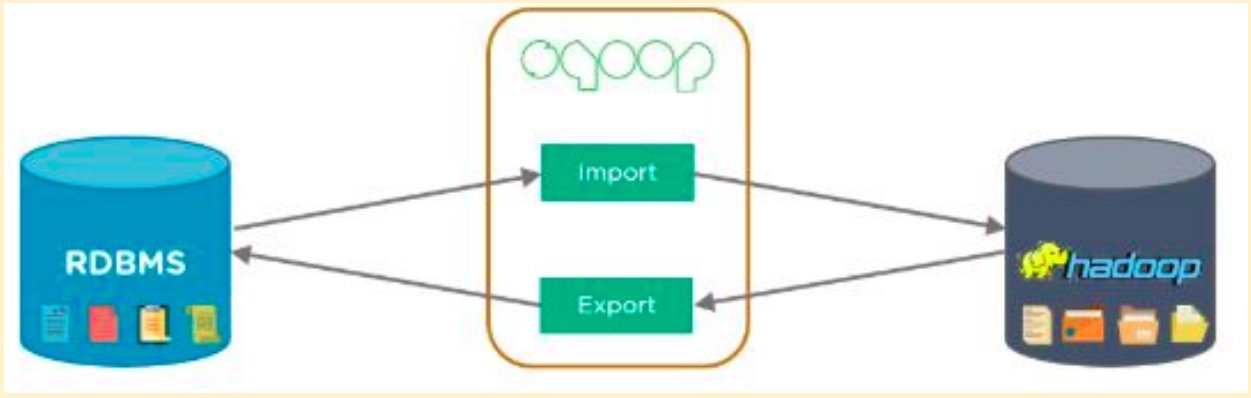
    - Linux/Mac -
        - Either install putty and connect **OR** run below command
        - Go to Terminal → ssh <user_name>@<remote_host>
            - For example - **ssh cloudera@hostIdAddress**

3. Run Command to leave safe mode - **hadoop dfsadmin --safemode leave**



SQOOP -

2009 cloud support - Kathleen Ting
Tool used to transfer bulk data between HDFS and RDBMS
Sqoop word came from SQL+HADOOP=SQOOP



To process data using Hadoop, the data first needs to be loaded into Hadoop clusters from several sources. However, it turned out that the process of loading data from several heterogeneous sources was extremely challenging.

Issued Big data developers faced with Hadoop -

1. RDBMS is tough to Handle
2. Damn Slow to import data from RDBS to Hadoop
3. Portion(partial) Data Import is not possible
4. Incremental Data import is not possible
5. Change of data is Hard
6. Serialized data has no support
7. Export no much Support etc

Doug Cutting → Thats not my problem

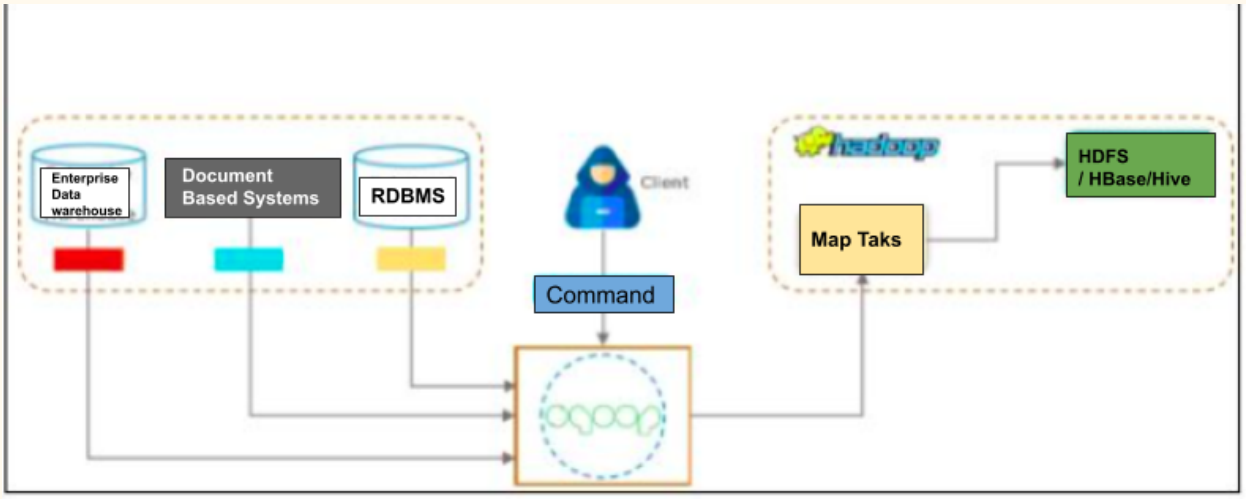
2008 → Data ingestion has become big problem

2009 → **Kathleen** -> I have a solution to Integrate SQL and Hadoop

1) RDBMS integration is Made Simple
2) My tool is damn faster (Multi threading)
3) Yes you can import certain portion of data happily
4) Obviously incremental data import is possible
5) Change of Data =- absolutely
6) My tool is made of Java (Serialization Support is good)
7) Exportsss is possible now.

SQOOP Architecture

Sqoop fetches data from different databases. Here, we have an enterprise data warehouse, document-based systems, and a relational database. We have a connector for each of these; connectors help to work with a range of accessible databases.



SQOOP Command

- sqoop import
  --connect jdbc:mysql://<hostname>:<portName>/<DBname>
  --username <userName>
  --password <password>

```
    --table <sourceTableName>
    --m 1
    --target-dir <targetDirectoryPath>
```

- **sqoop job --create <jabName** // Create Sqoop job
- **sqoop job --list** // To check all job list
- **sqoop job --exec <jobName>** // To run specific job
- **sqoop job --show <jobName>** // To check saved job details
- **sqoop job --delete <jabName>** // To delete job

Prerequisites -

Login into remote Gateway either from SSH or virtualBox cloudera [How to Login into Edge node/Gateway node]

For big data classes -

- **hostname** → localhost
- **portnumber** → 3306
- **database** → zeyodb
- **username** → root
- **password** --> Aditya908
- **table** → ztab
- **targetlocation** → /user/cloudera/dimport

For Big data practice class - how to run SQOOP

- **We are going to use cloudera in VirtualBox - Open Cloudera and login into MySQL to check if SQL is working**

  mysql --uroot --pcloudera and now run the below command in SQL

```
===========
Mysql comannds
===========

mysql -uroot -pcloudera
create database zdb;
use zdb;
create table zeyotab(id int,name varchar(100),city varchar(100));
insert into zeyotab values(1,'Sai','Chennai');
insert into zeyotab values(2,'Zeyo','Hyderabad');
insert into zeyotab values(3,'Analytics','Hyderabad');
insert into zeyotab values(4,'Sai','Chennai');
insert into zeyotab values(5,'Hema','Hyderabad');
insert into zeyotab values(6,'Vassu','Chennai');
select * from zeyotab;
quit
```

- **Open another Tab and Then run below command -**

```
sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password Aditya908 --table ztab --m 1 --target-dir /user/cloudera/dimport
```

==========
Edge Node
==========

- sqoop import --connect jdbc:mysql://localhost/zdb --username root --password cloudera --table zeyotab --m 1 --delete-target-dir --target-dir /user/cloudera/datadir
- hadoop fs -ls /user/cloudera/datadir

  Two files will be created like below - only part-m-0000 contains data

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/inimport
Found 3 items
-rw-r--r--   1 cloudera cloudera          0 2022-06-26 05:22 /user/cloudera/inimport/_SUCCESS
-rw-r--r--   1 cloudera cloudera         98 2022-06-26 05:22 /user/cloudera/inimport/part-m-00000
```

- hadoop fs -cat /user/cloudera/datadir/part-m-00000

Partial import data - We have 2 way

1. **--where** <condition>

```
sqoop import --connect jdbc:mysql://localhost/zdb --username root --password cloudera
--table zeyotab --where "city='chennai' and id>1"
 --m 1 --delete-target-dir --target-dir /user/cloudera/datadir
```

2. **--query -** for example let join two tables.
   - Important thing for command - this is **where \$CONDITIONS** compulsory to put wherever using **--query** command

```
sqoop import --connect jdbc:mysql://localhost/zdb1 --username root --password cloudera
--query "select a.*,b.product from zeyotab a join zeyoprod b on a.id=b.id where \$CONDITIONS"
--m 1 --delete-target-dir --target-dir /user/cloudera/joindata
```

   - even if there is any other condition like id>3, put is as - **where id>3 and \$CONDITIONS**

Increatmental import data -

```
sqoop import --connect jdbc:mysql://localhost/zdb2 --username root --password cloudera
--table zeyotab --m 1 --target-dir /user/cloudera/inimport
 --incremental append --check-column id --last-value 6
```

Argument	Description
--check-column (colName)	**Specifies the column to be examined when determining which rows to import.**
--incremental (mode)	**Specifies how Sqoop determines which rows are new. Legal values for mode include append and lastmodified.**
--last-value (value)	**Specifies the maximum value of the check column from the previous import.**

If providing these above 3 details, there will run 2 Map-reduce jobs -

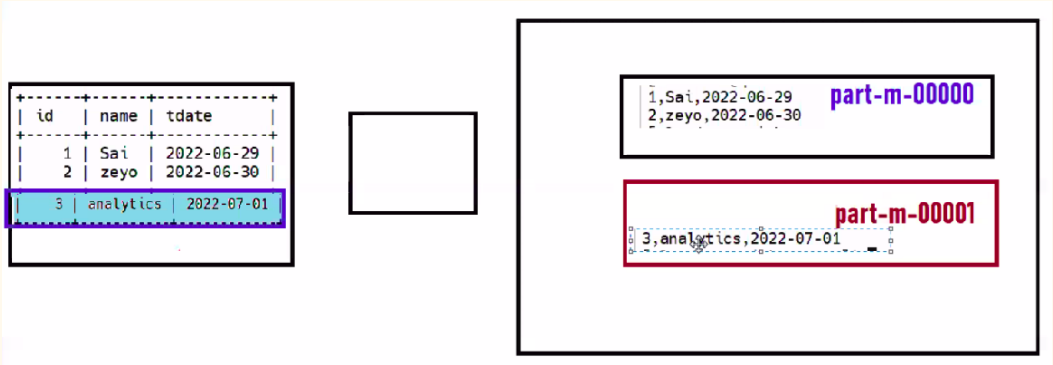
- To bring Data from RDBMS
- To override the data to HDFS

★ **Rules**

➔ Pick a column that should have only integer-like - numbers, dateTime

➔ That column shouldn't have duplicates otherwise increment process won't work for duplicates

➔ A new part file will be created for every increment data, Process -



```
============================
Mysql commands
============================
  •   mysql -uroot -pcloudera
  •   create database zdb2;
  •   use zdb2;
  •   create table zeyotab(id int ,name varchar(100),city varchar(100));
  •   insert into zeyotab values(1,'Sai','Chennai');
  •   insert into zeyotab values(2,'Zeyo','Chennai');
  •   insert into zeyotab values(3,'Analytics','Hyderabad');
  •   insert into zeyotab values(4,'Sai','Chennai');
  •   insert into zeyotab values(5,'Hema','Hyderabad');
  •   insert into zeyotab values(6,'Vassu','Chennai');
  •   select * from zeyotab;

========================
Edge Node
========================
  •   sqoop import --connect jdbc:mysql://localhost/zdb2 --username root --password cloudera --table zeyotab --m 1
      --delete-target-dir --target-dir /user/cloudera/inimport
  •   hadoop fs -ls  /user/cloudera/inimport
  •   hadoop fs -cat /user/cloudera/inimport/part-m-00000

============================
Again login mysql
============================
  •   mysql -uroot -pcloudera
  •   use zdb2;
  •   insert into zeyotab values(7,'Hema','Hyderabad');
  •   insert into zeyotab values(8,'Vassu','Chennai');
  •   select * from zeyotab;
  •   quit

==================================
Edge Node( for increament new part-m-00001 file will be created and will only contain increated data)
==================================

  •   sqoop import --connect jdbc:mysql://localhost/zdb2 --username root --password cloudera --table zeyotab --m 1
      --target-dir /user/cloudera/inimport --incremental append --check-column id --last-value 6
  •   hadoop fs -ls  /user/cloudera/inimport
  •   hadoop fs -cat /user/cloudera/inimport/part-m-00001
  •   hadoop fs -cat /user/cloudera/inimport/*
```

Updated/Modified import data

Sai Video - https://youtu.be/CGiJf7tAyIk

```
sqoop import --connect jdbc:mysql://localhost/zdb2 --username root
--password cloudera --table zeyotab --m 1 --target-dir /user/cloudera/inimport
--incremental lastmodified --check-column dataTime --last-value 6 --merge-key
id
```

Argument	Description
--check-column (colName)	Specifies the column to be examined when determining which rows to import.

--incremental (mode)	**Specifies how Sqoop determines which rows are new. Legal values for mode include append and lastmodified.**
--last-value (value)	**Specifies the maximum value of the check column from the previous import.**
--merge-key (colName)	**This column shouldn't have duplicates**

If providing these above 4 details, there will run 2 Map-reduce jobs -

- To bring Data from RDBMS
- To override the data to HDFS

One Case - For example, there are multiple part-m-0000 files in target dir, and after running the update/modified command, there will be only 1 part-m-00000 file and it will have all the data

Using the options file to Pass Arguments

For example, the following Sqoop comand for import can be specified alternatively as shown below:

```
sqoop import --connect jdbc:mysql://localhost/dbName --username foo –password cloudera --table TEST
```

TO

```
sqoop --options-file /users/homer/work/import.txt --table TEST --target-dir /user/cloudera/dataArgFile
```

★ where the options file /users/cloudera/work/import.txt contains the following: Make sure to provide a new line for every argument and value like below, also we can add more arguments in the file if want -

```
import
--connect
jdbc:mysql://localhost/db
--username
root
--password
cloudera
```

SQOOP-All-Tables-Import-

```
sqoop import-all-tables (generic-args) (import-args)
```

- If want to exclude few tables, run below →

```
sqoop import-all-tables  --connect --username --password cloudera
--exclude-tables Table498, Table 323, Table 199
```

SQOOP Job -

Question - Do we need to save the last value every time, it is hard to keep remembering the last value every time as data goes in bulk in a day many times?

Sai → kathleen Do i need keep remembering the last value everytime. as its very Tedious
Kathleen → Not required. There is a concept of Automation Jobs, Take your sqoop incremental import and come to me
Sai —>

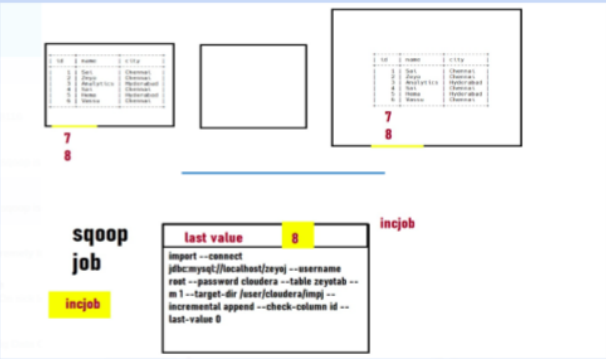
sqoop import --connect jdbc:mysql://localhost/zeyoj --username root --password cloudera --table zeyotab --m 1 --target-dir /user/cloudera/impj --incremental append --check-column id --last-value 0

Kathleen —> Automation job will have below role -

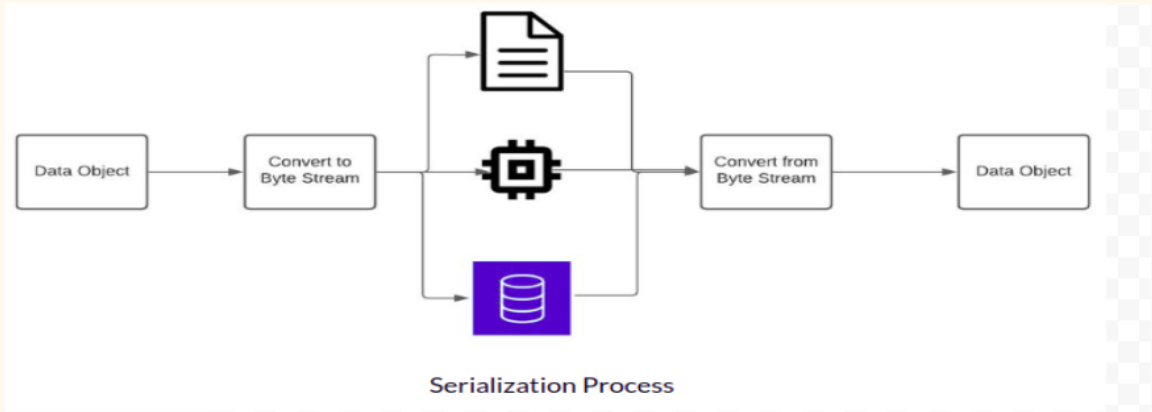
1. Send the incremental Import Command to a job and assign name to it , like → **increamentJob**

Steps

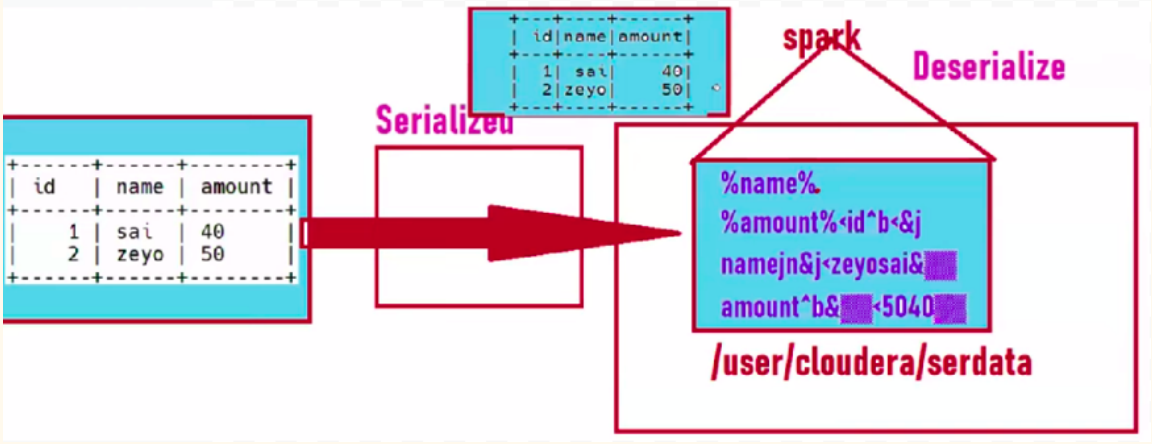
1. Create a job using below command -
   sqoop job --create increamentJob
2. Assign incremental command -
   sqoop job --create increamentJob -- import --connect jdbc:mysql://localhost/zeyoj --username root --password cloudera --table zeyotab --m 1 --target-dir /user/cloudera/impj --incremental append --check-column id --last-value 0
3. Check whether it got created -
   sqoop job --list
4. Execute the Job and check target -
   sqoop job --exec increamentJob
5. Insert some more records in sql.
6. Execute the Job and check target -
   sqoop job --exec increamentJob

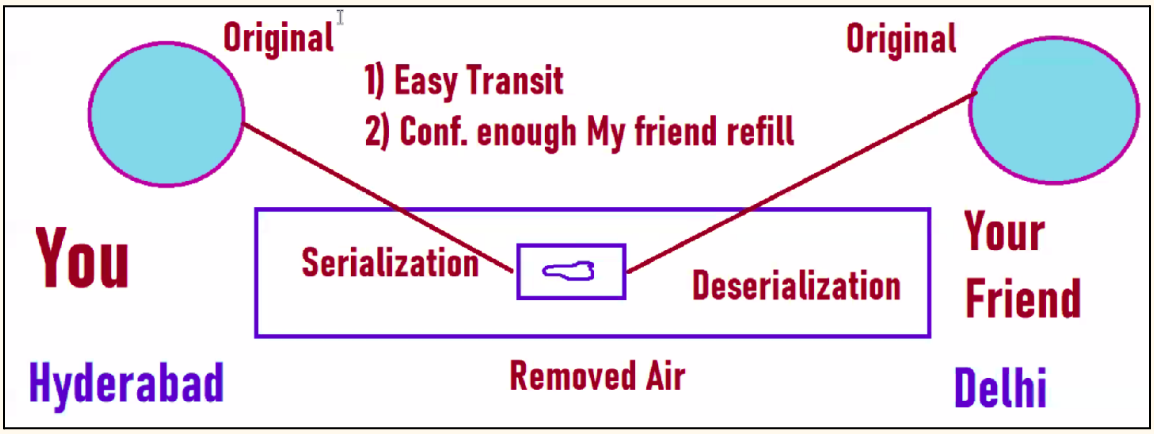


Serialization File Types



Serialization Process





Below are serialization file formats.

1. Text files

--as-textfile

Features -

- Huge in size
- Readable format

- If you query on TextFile is huge
- Very bad option for Big Data

2. Sequence data files - 2008

--as-sequencefile

```
sqoop import --connect jdbc:mysql://localhost/serdb --username root --password cloudera --table avtab --m 1 --delete-target-dir --target-dir  /user/cloudera/sequencedir
--as-sequencefile

hadoop fs -cat /user/cloudera/sequencedir/*
```

Features -

- Java File format - it is made of java and since Mapreduce is also Made Java it is very friendly
- Not much used Today as Mapreduce is not used Nowadays
- Sequence is also Huge in Size
- It's not good for Queries

3. Avro Data file - (2009)

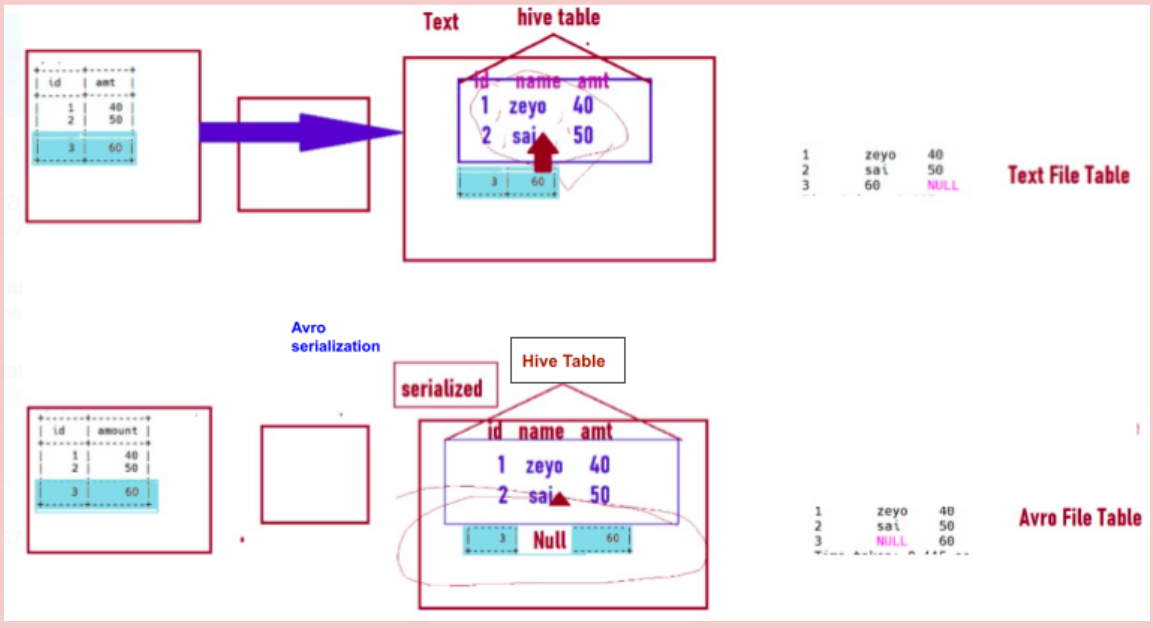
--as-avrodatafile

```
sqoop import --connect jdbc:mysql://localhost/serdb --username root --password cloudera --table avtab --m 1 --delete-target-dir --target-dir
/user/cloudera/avrodir --as-avrodatafile

hadoop fs -cat /user/cloudera/avrodir/*
```

Features -

- It can handle the Schema evolution, like deleting a column, adding a new column--- It can map data to right columns
- 40-50% Compression
- Row format storage

- Sai, I have table created in mysql, and Done very normal Import (text file) in HDFS.
- I checked the data its fine with textfile, To process the data I have created hive table also on top
- Queried the data, Data is also Good
- **But When the schema evolution (Schema changes -- Column changes like delete name Cloumn just like in below example)**
- Hive or mapreduce could not tackle that



==========
mysql
==========

- create database avrodb;
- use avrodb;
- create table avro_tab(id varchar(100),name varchar(100),amount varchar(100));
- insert into avro_tab values(1,'zeyo',40);
- insert into avro_tab values(2,'sai',50);

==========
Edge Node
==========

sqoop import --connect jdbc:mysql://localhost/avrodb --username root --password cloudera --table avro_tab --m 1 --target-dir /user/cloudera/avrodir --as-avrodatafile

==========
hive
==========

- hive
- CREATE EXTERNAL TABLE ahivetab ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED as AVRO LOCATION '/user/cloudera/avrodata' TBLPROPERTIES ('avro.schema.url'='/user/cloudera/avro_tab.avsc');
- select * from phivetab;

```
1       zeyo    40
2       sai     50
```

==========
mysql
==========

- alter table avro_tab drop column name;
- insert into avro_tab values(3, 50);

==========
Edge Node
==========

Now run increment command

- sqoop import --connect jdbc:mysql://localhost/avrodb --username root --password cloudera --table avro_tab --m 1 --target-dir /user/cloudera/avrodir --as-avrodatafile --increment append --check-column id --last value 2

==========
hive
==========

- select * from phivetab;

```
3       NULL    40
4       NULL    60
1       zeyo    40
2       sai     40
```

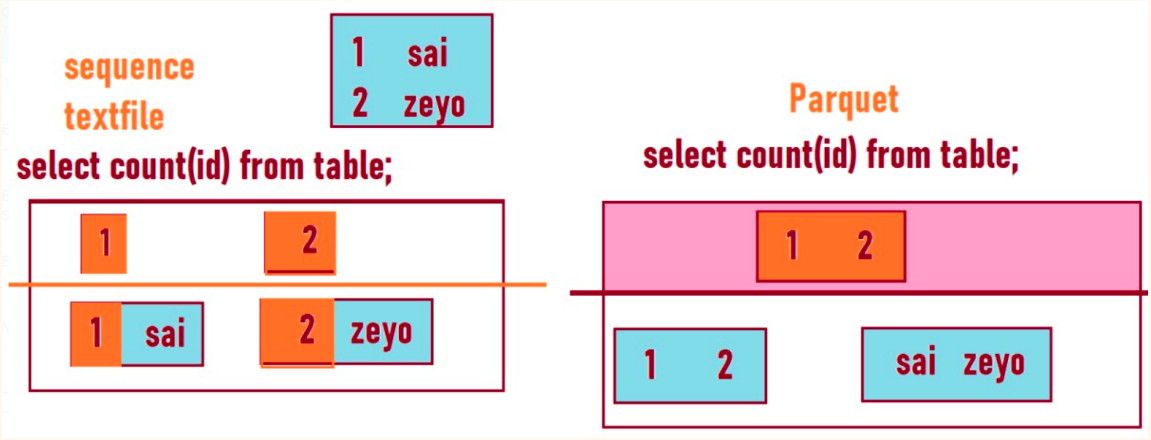
4. PARQUET - Came in 2012 by Doug Cutting, it is a very powerful file format yet

--as-parquetfile

```
sqoop import --connect jdbc:mysql://localhost/serdb --username root --password cloudera --table avtab --m 1 --delete-target-dir --target-dir /user/cloudera/parquetdir --as-parquetfile

hadoop fs -cat /user/cloudera/parquetdir/*
```

Features -

i. Reduce compression from 60-80%
ii. It saves the file data in **columnar file format**
iii. Faster query file format
iv. Parquet uses Predicate pushdown mechanism



==========
mysql
==========

- create database pardb;
- use avrodb;
- create table par_tab(id varchar(100),name varchar(100),amount varchar(100));
- insert into par_tab values(1,'zeyo',40);
- insert into par_tab values(2,'sai',50);

==========
Edge Node
==========
sqoop import --connect jdbc:mysql://localhost/pardb --username root --password cloudera --table par_tab --m 1 --target-dir /user/cloudera/pdir --as-parquetfile
==========
hive
==========

- hive
- CREATE TABLE phivetab (id string, name STRING,amount string) STORED AS PARQUET location '/user/cloudera/pdir';
- select * from phivetab;

1	zeyo	40
2	sai	40

==========
mysql
==========

- alter table par_tab drop column name;
- insert into par_tab values(3, 50);

==========
Edge Node
==========
Now run increment command

- sqoop import --connect jdbc:mysql://localhost/pardb --username root --password cloudera --table par_tab --m 1 --target-dir /user/cloudera/pdir --as-parquetfile --increment append --check-column id --last value 2

==========
hive
==========

- select * from phivetab;

3	NULL	40
4	NULL	60
1	zeyo	40
2	sai	40

5. ORC (Optimized Row Columnar)-

We can not do SQOOP import for ORC

Features -

- Compression rate is 90-95%
- Query takes time because de-compression takes time
- Historical Data
- Columnar File
- Faster but not much as Parquet file

Tasks/Interview questions

#1 - Find a way to import only Chennai records, But I need only id and name column

sqoop import --connect jdbc:mysql://localhost/zdb --username root --password cloudera --table zeyotab --where "city='chennai' and id>1"
--columns id,name
--m 1 --delete-target-dir --target-dir /user/cloudera/datadir

#2 - Now I have created Automation jobs but i have do enter Password every time, it is too much manual work?

To reduce this work, create Password file and provide path of file in sqoop command

Create file either in Edge node or HDFS node - ***(but one condition there should be no new line character in file)***
echo -n cloudera>spfile

SQOOP command -

sqoop job --create increamentJob -- import --connect jdbc:mysql://localhost/zeyodb6 --username root --password-file file:///home/cloudera/spfile
--table zeyod --m 1 --target-dir /user/cloudera/djimport --incremental append --check-column id --last-value 0

For HDFS node provide file like this - **hdfs:/user/cloudera/spfile**

#3 - Where does all jobs data is saved like last value , upper value and everthing -

There is file name metastore.db.script under .sqoop directory

1. **ls .sqoop**

```
[cloudera@quickstart ~]$ ls .sqoop
metastore.db.properties  metastore.db.script
[cloudera@quickstart ~]$
```

2. **cat metastore.db.script**

#4 - How can we import data from particular row or column? What is the destination types allowed in Sqoop import command?

- sqoop import –connect jdbc:mysql://db.one.com/corp --table INTELLIPAAT_EMP **--where "start_date> '2016-07-20' "**
- Sqoop import --connect jdbc:mysql://db.test.com/corp **--query "SELECT * FROM intellipaat_emp LIMIT 20"**
- sqoop import –connect jdbc:mysql://localhost/database --username root --password aaaaa **--columns "name,emp_id,jobtitle"**

#5 - I am getting connection failure exception during connecting to Mysql through Sqoop, what is the root cause and fix for this error scenario?

This will happen when there is lack of permissions to access our Mysql database over the network

Go to MySql and run below command -

```
mysql> GRANT ALL PRIVILEGES ON *.* TO ''@'localhost';
```

#6 I am having around 500 tables in a database. I want to import all the tables from the database except the tables named Table 498, Table 323, and Table 199. How can we do this without having to import the tables one by one?

sqoop **import-all-tables** --connect –username --password **--exclude-tables Table498, Table 323, Table 199**

Multi-Mappers -

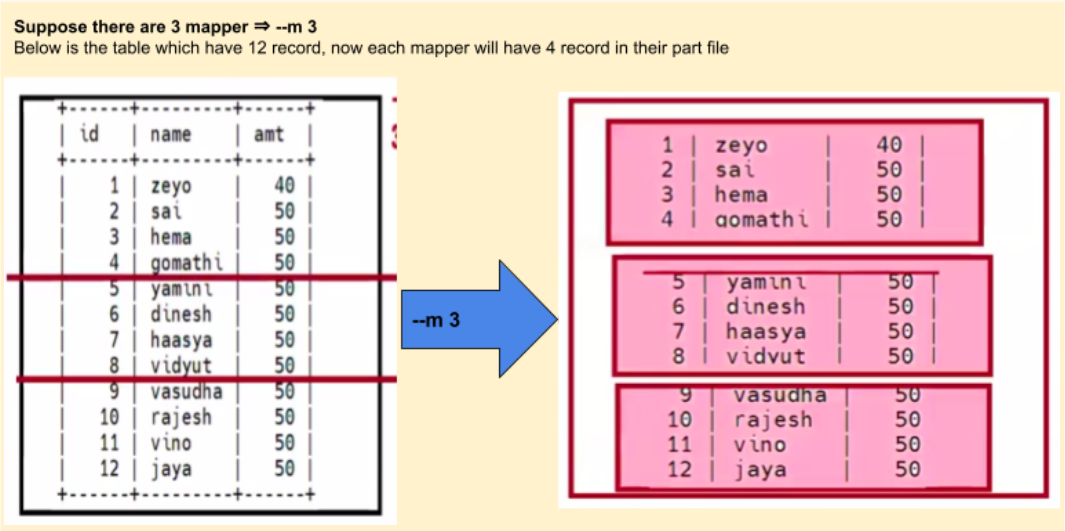
Multi-mapper is nothing but multi-threading. (Arg in sqoop ⇒ **--m 1**).

- We use multi-mapper to make imports faster
- If you want to increase mapper value(**-- m 2)** then you have to provide column value by adding an argument in command **--split-by <columnName>** or there should be a primary key in the table. If not provide any of these values then there will be an error like the below -

```
22/07/10 09:11:00 ERROR tool.ImportTool: Import failed: No primary key could be found for table avro_tab. Please specify one
with --split-by or perform a sequential import with '-m 1'.
```

- By default, sqoop export uses 4 (**--m 4**)threads or number of mappers to export the data

- For each mapper one separate part is created.



- Mappers divide rows between multi-mappers using MIN and MAX values.

Performance Tuning

In General, performance tuning in Sqoop can be achieved by:

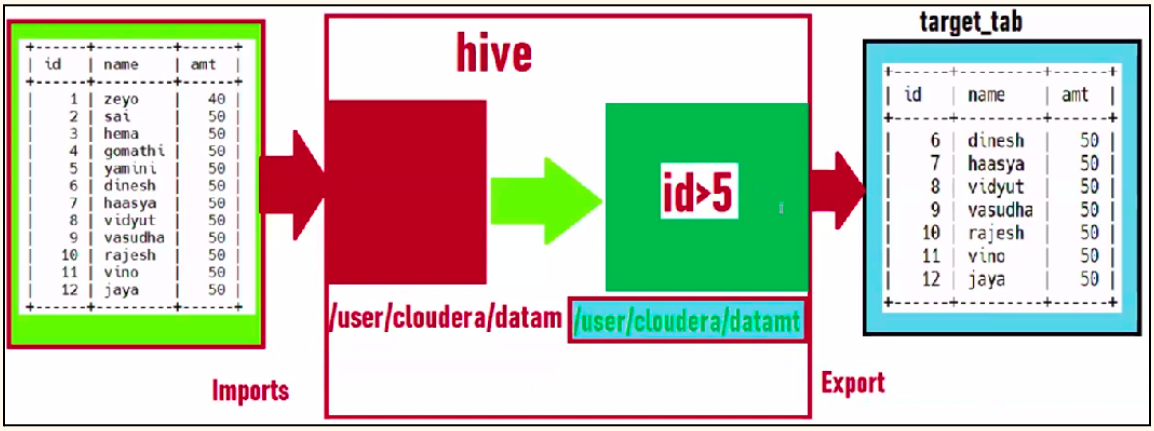
- Controlling Parallelism
  - Using Mappers
- Controlling Data Transfer Processes
  - Batching
  - Direct

Export in SQOOP -

Using SQOOP Export data to MySQL from HDFS

```
SQOOP EXPORT --connect jdbc://localhost/<dbName> --username root --password cloudera --m 1 \
--table <targetTableName> --staging-tab <stagingTable> --export-dir <dirPath>
```

*Note - Here providing staging table(**--stage-tab**) role is "**Data will export to Stage table first then it will go to Target_table",** so that if the process couldn't be completed due to any reason, we can easily delete data from the staged table.



Using SQOOP move HDFS data to Hive

Using SQOOP move MySql data to Hive

Locations in Hive

Cloud integration with SQOOP -

Command - Requires 3 arguments -

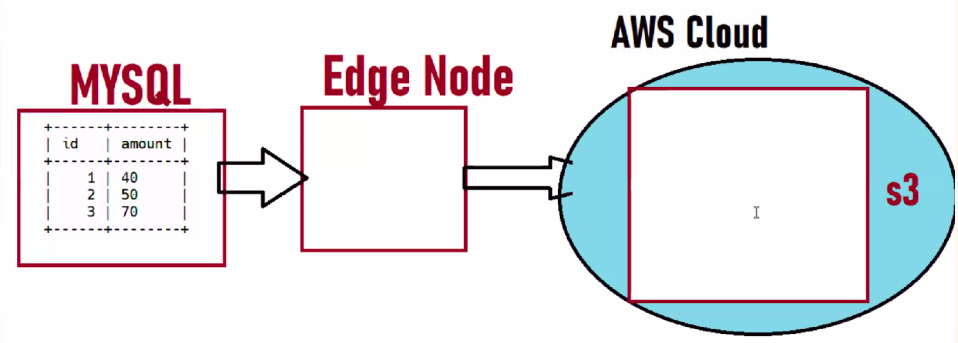
```
sqoop import
--Dfs.s3a.access.key=<accessKey>
--Dfs.s3a.secret.key=<SecertKey>
--Dfs.s3a.endpoint=<AWS-s3-endpiont>
--connect jdbc:mysql://localhost/dd2 --username root --password cloudera --table ztab
--m 1 --target-dir  s3a://zeyobuck/zeyosqoop/Datadir
```

==========
mysql
==========

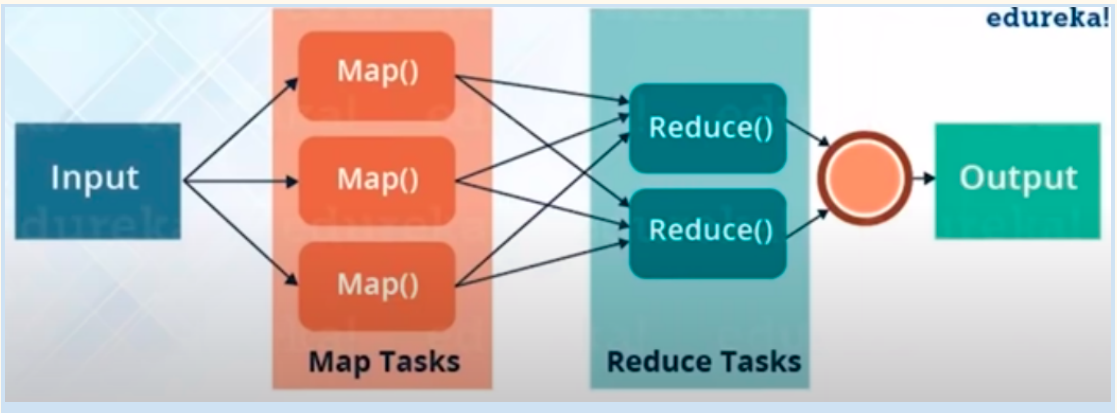
- mysql -uroot -pcloudera
- create database dd2;
- use dd2;
- create table ztab(id int,name varchar(100),tdate date);
- insert into ztab values(1,'Sai', now() - interval 3 day);
- insert into ztab values(2,'zeyo', now() - interval 2 day);
- quit

====================
Edge Node
====================
sqoop import
-Dfs.s3a.access.key=AKIAUDT4DPHYY3EHGND3 -Dfs.s3a.secret.key=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
-Dfs.s3a.endpoint=s3.ap-south-1.amazonaws.com
--connect jdbc:mysql://localhost/dd2 --username root --password cloudera --table ztab --m 1 --target-dir s3a://zeyobuck/zeyosqoop/<URNAME>dir

- To verify folder is created in AWS, run below command
  - cd .aws
  - wget https://zeyobuck.s3.ap-south-1.amazonaws.com/credentials
  - cd
  - aws s3 ls s3://zeyobuck/zeyosqoop/



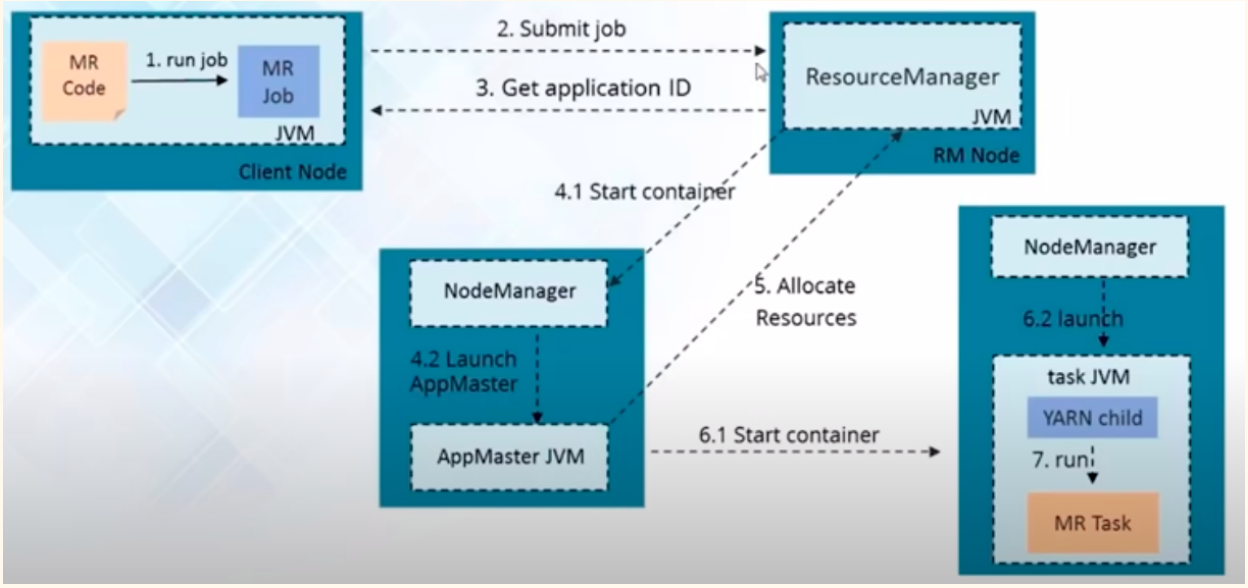
MapReduce



Components of MapReduce

1. **PayLoad**: The applications implement Map and Reduce functions and form the core of the job
2. **MRUnit**: Unit test framework for MapReduce
3. **Mapper**: Mapper maps the input key/value pairs to the set of intermediate key/value pairs
4. **NameNode**: Node that manages the HDFS is known as namednode
5. **DataNode**: Node where the data is presented before processing takes place
6. **MasterNode**: Node where the job trackers runs and accept the job request from the clients
7. **SlaveNode**: Node where the Map and Reduce program runs
8. **JobTracker**: Schedules jobs and tracks the assigned jobs to the task tracker
9. **TaskTracker**: Tracks the task and updates the status to the job tracker
10. **Job**: A program that is an execution of a Mapper and Reducer across a dataset
11. **Task**: An execution of Mapper and Reducer on a piece of data
12. **Task Attempt:** A particular instance of an attempt to execute a task on a SlaveNode

MapReduce Job workflow -



Abstraction of MapReduce -

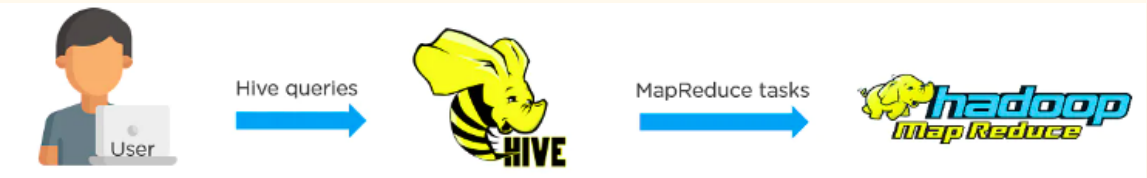
1. Hive - Query engine(uses SQL)
2. Pig - yahoo invented
3. Sqoop

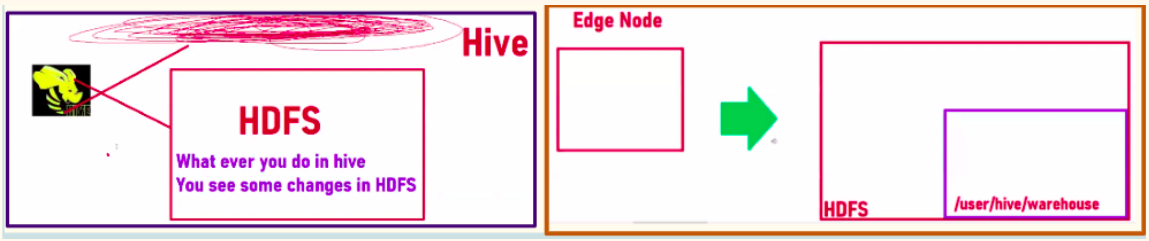
4. Oozie - yahoo invented use for automating/scheduling

HIVE -

Developed by JoyDeep Sen Sarma.

- **Hive runs on top of HDFS. In simple, whatever we do in Hive there will be the same changes save in HDFS (dir ⇒ /user/hive/warehouse/)**





- Hive table should always be pointed to a location. While creating a table, please ensure to provide a directory Path to the table. If you do not specify any pointing directory, Hive will create a directory in the table's name and get pointed to it.
    - Check table Location - **describe formatted <tableName>**

```
# Detailed Table Information
Database:               hivedb
Owner:                  cloudera
CreateTime:             Sun Jul 17 02:58:46 PDT 2022
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/hivedb.db/hivetab
```

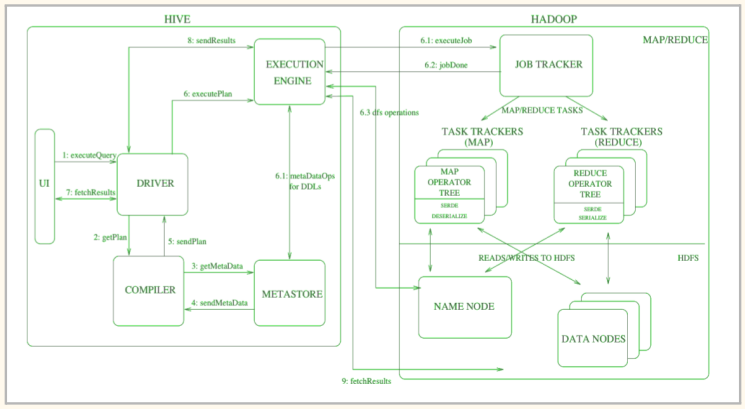
    - Location **/user/hive/warehouse/** is the default hive location to store all tables and its data if the location is not provided. To provide Location, Read - [Locations in Hive]
- **Apache Hive is a data warehouse system built on top of Hadoop and is used for analyzing structured and semi-structured data. Hive abstracts the complexity of Hadoop MapReduce. Basically, it provides a mechanism to project structure onto**

the data and perform queries written in HQL (Hive Query Language) that are similar to SQL statements. Internally, these queries or HQL get converted to map reduce jobs by the Hive compiler. Therefore, you don't need to worry about writing complex MapReduce programs to process your data using Hadoop. It is targeted toward users who are comfortable with SQL.

Why did HIVE come?

- ➔ As Map-Reduce is made of JAVA. Earlier Map-Reduce code was difficult to understand because java code wasn't easy at that time.
- ➔ To process data in HDFS from one directory to another, you have to follow the below steps -
  - ◆ Should knowledge of JAVA
  - ◆ Should knowledge of Map-Reduce code
  - ◆ Do Tune Mappers
  - ◆ Do Tune Reducers
  - ◆ And then export it in JAR
  - ◆ Take that JAR to cluster and deploy it
- ➔ Then the Hive tool comes to eliminate the use of Map-Reduce, For HIve, you don't need any Java or Map-Reduce code (although in the backend it uses Map-Reduce).
- ➔ Hive looks like SQL but not SQL
- ➔ Easy to Work with it
- ➔ **Hive internally uses MapReduce to process queries.**
- ➔ Hive only replaces JAVA use, not MapReduce.
- ➔ Hive can on the top these 3 engines-
  - ◆ MapReduce
  - ◆ Tez
  - ◆ Spark

Hive Architecture

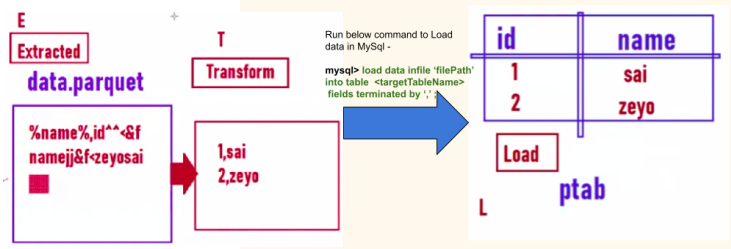


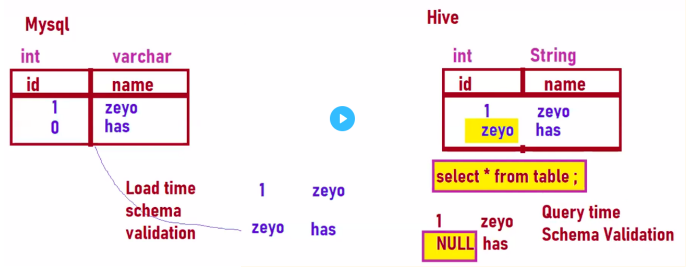
Once Hive query is triggered. First Driver takes the query and send to compiler to check syntax and semantic checks and sends to the execution Engine.

Then execution Engine query the data from Hadoop using Yarn and show you the data

HIVE vs MySql

MySql	HIVE
Data storage for MySql is DATABASE	Data storage for HIVE is HDFS
E T L (Extracted Transform Load) - E ⇒ T ⇒ L **Extracted** - Extract de-serialize data **Transform** - Convert it into Serialized data **Load** - Then Load it in SQL Run below command to Load data in MySql - **mysql>** `load data infile 'filePath' into table <targetTableName> fields terminated by ',' ;`	**E L T** (Extracted Load Transform) - E ⇒ L ⇒ T In Hive we just need to Extract data and run load command, Transform will take care By **HIVE.** **Load** - To Load data in Hive, Run below command to Load data in MySql - **hive>** `load data local inpath /user/cloudera/data.parquet into table '<targetTableName>';`

	
Good for Small Data, for large data it is costly like server, nodes, set up, installation, license	Good and free for large data

Load time schema validation (Schema on write)- once inserting data and changing the data type , data will set to 0 at load time.  ========== **Edge Node** ========== vi schemaCheck - 1,zeyo 2,bron zeyi,bron bron,4 ========== **SQL Node** ========== Now load 'schemaCheck' file in SQL • **load data local infile '/home/cloudera/sck' into table scktab fields terminated by ',';** • **SELECT * from scktab;** +------+------+ | id | name | +------+------+ | 1 | zeyo | | 2 | bron | | 0 | bron | | 0 | 4 | +------+------+ • For example if have provide string value in id column like in 'sechmaCheck' file, but in table **ID** column is only accepting integer, so for these value in SQL value is set to to 0 in table - **it means data changes at LOAD TIME SCHEMA**	**Query time schema validation (Schema on Read)**- once inserting data and changing the data type , there will be no change in data, But once you query the data it will give you NULL. But originally at backend data have same value that was provided during insert query. Now after load and query data, get the location of table and check original data at backend - $ **describe formatted <tableName>;** ⇒ get Hive file location $ **hadoop fs -cat /hive/cloudera/<dbName>/<tabName>/*** ========== **Edge Node** ========== vi schemaCheck 1,zeyo 2,bron zeyi,bron bron,4 ========== **HIVE Node** ========== Now load 'schemaCheck' file in SQL • **CREATE TABLE scktab(id int, name string) row format delimited fields terminated by ',' ;** • **load data local inpath '/home/cloudera/sck' into table scktab ;** • **SELECT * from scktab;** For example if have provide string value in id column like in 'sechmaCheck' file, but in table **ID** column is only accepting integer, But once you QUERY- mean SELECT data from table, for these value in HIVE is set to **NULL** in table - 1 zeyo 2 bron NULL bron NULL 4 • **describe formatted <tableName>;** ⇒ get Hive file location # Detailed Table Information Database: hivedb Owner: cloudera CreateTime: Sun Jul 17 11:35:05 PDT 2022 LastAccessTime: UNKNOWN Protect Mode: None Retention: 0 Location: hdfs://quickstart.cloudera:8020/user/hive/warehouse/hivedb.db/scktab Table Type: MANAGED_TABLE • **hadoop fs -cat /hive/cloudera/<dbName>/<tabName>/*** But if you check at **BACKEND** data is same as it was stored originally - **it means data changes at QUERY TIME** [cloudera@quickstart ~]$ hadoop fs -cat /user/hive/warehouse/hivedb.db/scktab/* 1,zeyo 2,bron zeyi,bron bron,4 [cloudera@quickstart ~]$
SQL only query **STRUCTURED** data	Hive can query **STRUCTURED, Semi-STRUCTURED** data
SQL don't have capability to query large data	Hive have capacity to only query PetaByte data
No serialization data support	Good serialization data support

Licenced	Free

Hive Warehouse

Hive warehouse is like hive's personal room for creation of Directories and Table. If we create hive table without location the directory gets created in the name of the table and my table gets pointed to it and we also create hive table with location on top of existing directory.

Type of Table in Hive -

1. MANAGED/Intetowardsrnal Table -

An *internal table* is a table that Hive manages. If you delete an internal table, both the definition in Hive *and* the data directory are deleted inside HDFS.

```
CREATE TABLE IF NOT EXISTS Cars(Name STRING,  id INT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
   LOCATION '/user/<username>/dataDir;

Drop table Cars;
```

- **Now if you Go to the location and check, directory won't be there anymore**
- Table created for Intermediate data processing, should be Managed type. So we can drop the table once the process is done.

2. EXTERNAL Table

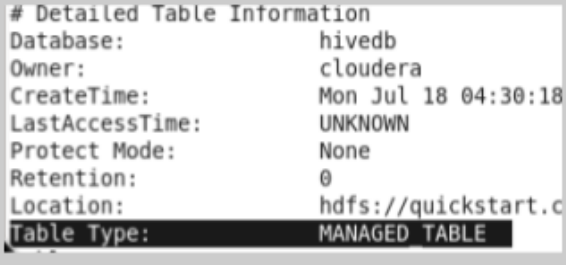
An *external table* is a table for which Hive does not manage storage. If you delete an external table in Hive, only the definition in Hive is deleted. The data directory remains inside HDFS.

```
• CREATE EXTERNAL TABLE IF NOT EXISTS Cars(Name STRING, id INT)
      COMMENT 'Data about cars from a public database'
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      STORED AS TEXTFILE
     LOCATION '/user/<username>/dataDir;

• Drop table Cars;
```

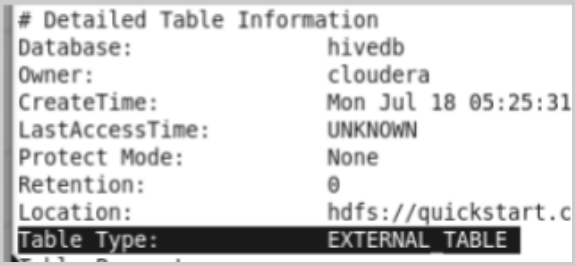
- **Now if you Go to the location and check, the directory will be presented there**
- Target table should always be EXTERNAL. Suppose if the table is deleted by mistake, it shouldn't impact on target analytics data.

Question #1 - How to check if table is INTERNAL or EXTERNAL?

- DESCRIBE FORMATTED <tableName>



```
# Detailed Table Information
Database:               hivedb
Owner:                  cloudera
CreateTime:             Mon Jul 18 04:30:18
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.c
Table Type:             MANAGED_TABLE
```



```
# Detailed Table Information
Database:               hivedb
Owner:                  cloudera
CreateTime:             Mon Jul 18 05:25:31
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.c
Table Type:             EXTERNAL_TABLE
```

Ques - What if we want to delete external table schema as well as data that exist in HDFS?
Yes, it is possible but need to configure table with `TBLPROPERTIES ('external.table.purge'='true');`

```
CREATE EXTERNAL TABLE IF NOT EXISTS names_text(a INT, b STRING) ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/andrena'
      TBLPROPERTIES ('external.table.purge'='true');
```

Load data from Edge Node into Hive

1. Create data file (text or .csv, etc) in EDGE node

```
2. CREATE TABLE table1(id int, name string) row format delimited fields terminated
by ',';
3. LOAD DATA LOCAL  inpath '/home/cloudera/<filePath>' into table <tableName>;
```

Load data from HDFS Node into Hive

1. Create data file (text or .csv, etc) in EDGE node

```
2. Put data in HDFS - hadoop fs -put datacsv.csv /user/cloudera/datafiles
3. CREATE TABLE tab(id int, name string) row format delimited fields terminated by
   ',';
4. LOAD DATA inpath '/home/cloudera/<filePath>' into table <tableName>;
```

Note - In step-4 after running the command ⇒ data file will be moved to Hive location (/user/hive/warehouse/<dbName>.db/<tabName>) and CSV or text file will be removed from HDFS location. But once loading a file from EDGE, the data file will be copied instead of completely moving, so the data file remains at both locations EDGE nodes and /user/hive/warehouse/...

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/hivedb.db/stab;
Found 2 items
-rwxrwxrwx   1 cloudera supergroup         25 2022-07-18 02:20 /user/hive/warehouse/hivedb.db/stab/datacsv.csv
-rw-r--r--   1 cloudera cloudera           24 2022-07-18 02:44 /user/hive/warehouse/hivedb.db/stab/datacsv_copy_1.csv
```

Requirement #1 - Find a way to insert the above source tab data into another Hive table

```
5. CREATE TABLE targetTab(id int, name string) row format delimited fields
   terminated by ',';
6. INSERT INTO targetTab select * from tab where <condition>;
```

Import MYSQL data to Hive using SQOOP

To import data into Hive, below arguments are important -

```
hive-import
--create-hive-table
--hive-table HiveDatabase.hiveTableName
```

Sqoop Command Option	Description
`--hive-import`	Imports tables into Hive using Hive's default delimiters if none are explicitly set.
`--hive-overwrite`	Overwrites existing data in the Hive table.

`--create-hive-table`	Creates a hive table during the operation. If this option is set and the Hive table already exists, the job will fail. Set to `false` by default.
`--hive-table <table_name>`	Specifies the table name to use when importing data into Hive.

If the database name is not provided in this argument --hive-table <tableName> then All data or tables which are imported from MySql will store in the DEFAULT database of Hive. You can find a newly created table under the Default database.

1. Import table into Default database -

Below command will import MySql table to Hive into Default database

- sqoop import --connect jdbc:mysql://localhost/<dbName> --username root --password cloudera --table <tableName> --m 1 --hive-import

2. Create table in hive then import SQL table data

```
sqoop import --connect jdbc:mysql://localhost/<dbName> --username root
--password cloudera --table <SqlTableName> --m 1
--hive-import --create-hive-table --hive-table hiveDB.<hiveTableName>
```

Note - A data file will be created under '/user/hive/warehouse/' in the name of the Hive table.

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/
Found 3 items
drwxrwxrwx   - cloudera supergroup          0 2022-07-18 06:56 /user/hive/warehouse/hivedb.db
drwxrwxrwx   - cloudera supergroup          0 2022-07-18 07:04 /user/hive/warehouse/tabhive2
```

3. We can import partition data, and incremental data as We do in SQOOP, just need to add the below values -

--hive-import --create-hive-table --hive-table <hiveTableName>

Import HDFS data to Hive using SQOOP

Below command will first ingest Data from Mysql to HDFS ⇒ Then created **process-m-00000 file** ⇒ will process to Hive table

```
sqoop import --connect jdbc:mysql://localhost:3306/sqoop --username root --password cloudera --m 1 --table customer
--target-dir /user/cloudera/sqlToHdfsDir
--fields-terminated-by ","
--hive-import
--create-hive-table
--hive-table sqoop_workspace.customers
```

Export Hive data to RDBMS using SQOOP

```
sqoop export --connect jdbc:mysql://localhost:3306/sqoop --username root --password cloudera --m 1 --table customer
--export-dir /user/hive/warehouse/test.db/demo
```

Locations in Hive

Suppose we are processing data like that -> HDFS ⇒ Source Table ⇒ Target Table

Requirement #1 - Find a way to process only specific data from HDFS to Target table but ensure do not use Source Table(As you did here - Requirement #1 - Find a way to insert above tab data into another Hive table)

1. While creating the table provide the location directory where your data is stored, and once you query the table it will show you data.

```
CREATE EXTERNAL TABLE IF NOT EXISTS Cars(Name STRING,  id INT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    LOCATION '/user/<username>/dataDir;
```

2. And once you query the table it will show you data. **SELECT * from cars;**

Question #1- What If we increase data at the source file end, will Increased rows show at the Hive end? ⇒ Yes

- Before data file have only 3 rows -

```
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/datafiles/*
4,rubal
5,rubal
6,rubal
```

- Added 2 more rows and new rows are showing at Hive end -

```
hive> select * from ttab;
OK
4       rubal
5       rubal
6       rubal
7       ayushi
8       zeyo
```

Question #2- What If we insert new Row in Hive table using INSERT query, will increased data reflect at Source file Location? ⇒ Yes

- Inserted data in HIve table - `hive> insert into ttab values(9, 'bron');`
- Checked at file location if new row is added there or not -

```
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/datafiles/*
9,bron
4,rubal
5,rubal
6,rubal
7,ayushi
8,zeyo
```

Hive File Formats

1. Load Text Data to Hive

```
create table avroTab(id int, name string) row format delimited fields terminated by ',' LOCATION '/user/cloudera/project1';
```

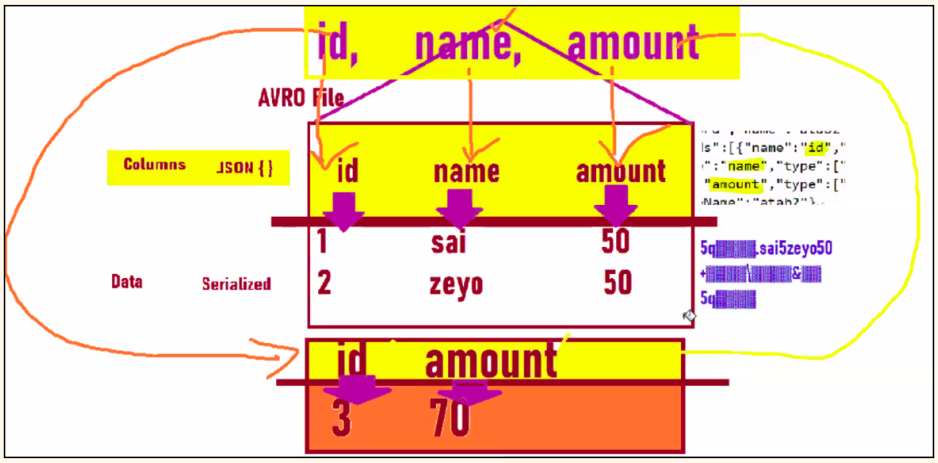
2. Load AVRO Data to Hive

```
create table avroTab row format
SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED AS AVRO LOCATION '/user/cloudera/project1'
TBLPROPERTIES ('avro.schema.url'='/user/cloudera/customer.avsc');
```

For more Practice refer Phase 1 - Project

- **AVRO supports schema evolution**
- **AVRO file has two sections ⇒ JSON schema and serialized data**
- **Since data has columns coupled, since which columns we call it gets fetched from File**

- When you create a hive table for avro data, you cannot specify column names. Instead you need AVSC file to be available



3. Load Parquet Data to Hive

- sqoop import --connect jdbc:mysql://localhost/pdb1 --username root --password cloudera --table par_tab --m 1 --target-dir /user/cloudera/pdir --as-parquetfile
- CREATE TABLE hiveTab (id STRING, name STRING) STORED AS PARQUET location '/user/cloudera/pdir';

4. Load ORC Data to Hive

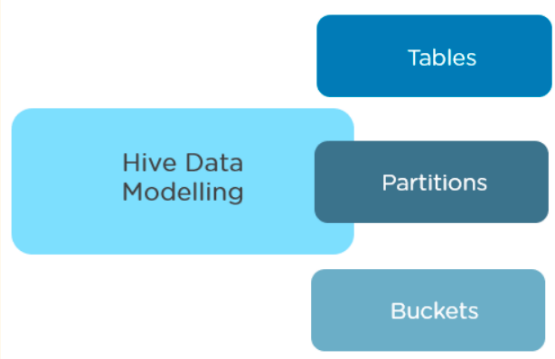
We can not upload data in ORC table using 'load data inpath' (because there is no SQOOP import for ORC). If we want to load data in ORC table we have to use the 'insert into or insert overwrite' command.

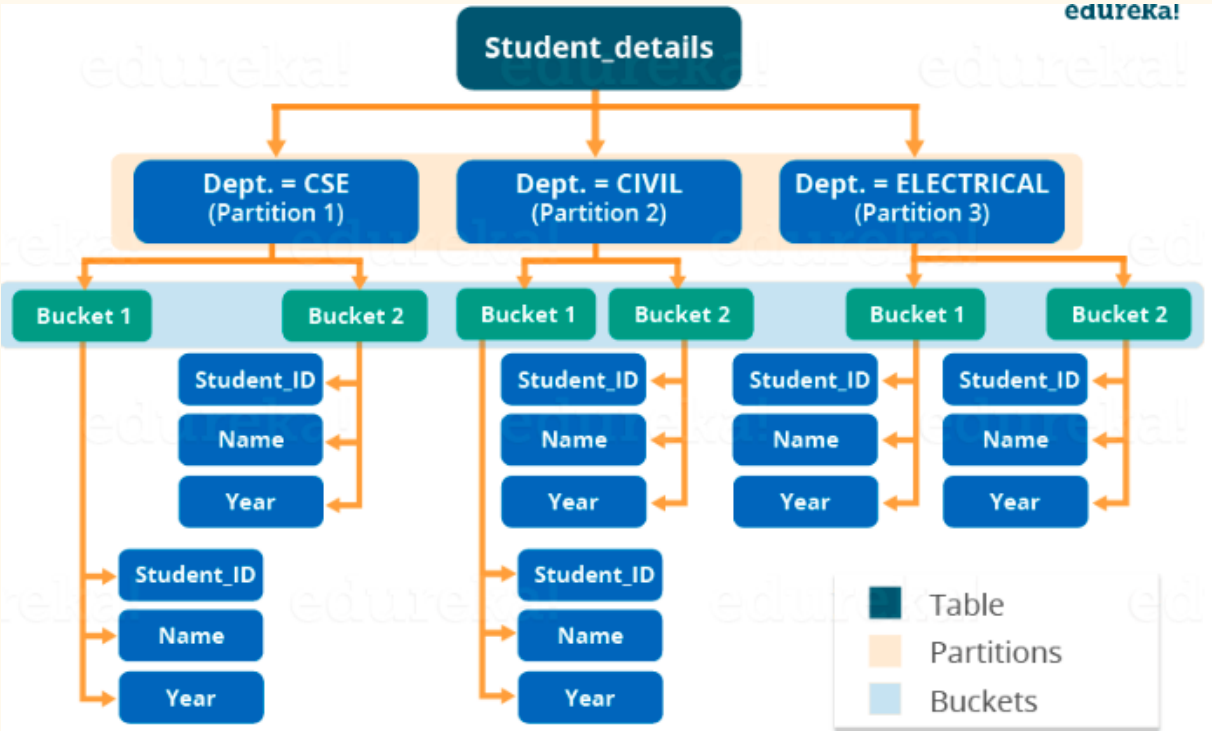
```
1. CREATE TABLE IF NOT EXISTS mycars(id INT, Name STRING)
       ROW FORMAT DELIMITED
       FIELDS TERMINATED BY ','
       STORED AS ORC;
2. INSERT INTO orcTab SELECT * FROM targetTab;
   ——————— or ———————
   INSERT OVERWRITE orcTab SELECT * FROM targetTab;
```

Hive Data Modeling

That was how data flows in the Hive. Let's now take a look at Hive data modeling, which consists of tables, partitions, and buckets:

1. Tables - Tables in Hive are created the same way it is done in RDBMS
2. Partitions - Here, tables are organized into partitions for grouping similar types of data based on partition key
3. Buckets - Data present in partitions can be further divided into buckets for efficient querying

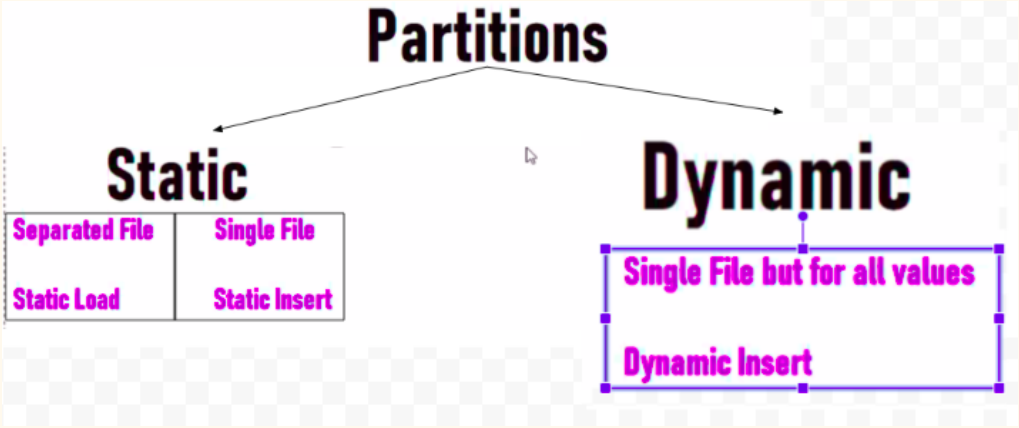




Partitions -

- Hive supports partitions by dividing the data into multiple folders or sub folders for efficient querying. Partition is good when data is low cardinal.
- Hive create default partition of 100. To increase these number run command -

  `hive.exec.max.dynamic.partitions`



1. Static Partition -

a. Static Load -

```
1. CREATE TABLE slPart(id int, name string)
   PARTITIONED BY(country string)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
2. load data local inpath '/home/cloudera/partdata/IND.csv' INTO table slPart
   PARTITION(country='IND');
```

#Scenario - We have data separated data in individual files for example country-wise(IND, UK, USA. Create Partitions Table and load each file individual.

IND.csv

```
1,Sai,I
2,zeyo,I
3,ze,I
4,bron,I
```

UK.csv

```
5,Hema,K
6,Gomathi,K
7,Ayushi,K
8,Ayu,K
```

US.csv

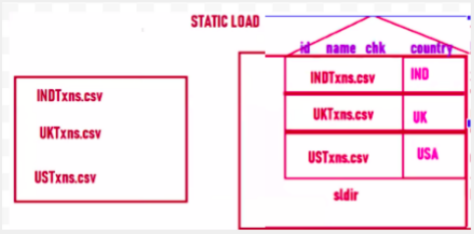
```
9,Jai,S
10,Swathi,S
11,Gupta,S
```

1. Create Static Partition Table -
   CREATE TABLE slPart(id int, name string, countryCode string) PARTITIONED BY(country string) \
   row format delimited fields terminated by ',' location '/user/cloudera/slPartDir';

2. Load each file in table and create individual Partition
   load data local inpath '/home/cloudera/partdata/IND.csv' into table slPart PARTITION(country='IND');
   load data local inpath '/home/cloudera/partdata/UK.csv' into table slPart PARTITION(country=US);
   load data local inpath '/home/cloudera/partdata/US.csv' into table slPart PARTITION(country=USA);

3. Go to HDFS directory and check partition files are created -

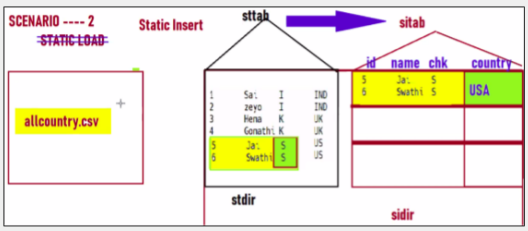
```
[cloudera@quickstart partdata]$ hadoop fs -ls /user/cloudera/slPartDir
Found 3 items
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 00:49 /user/cloudera/slPartDir/country=IND
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 00:54 /user/cloudera/slPartDir/country=UK
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 00:54 /user/cloudera/slPartDir/country=US
```



b. Static Insert -

```
1. CREATE TABLE siPartTable(id int, name string) PARTITIONED BY(country string) ROW FORMAT DELIMITED FIELDS
   TERMINATED BY ',';
2. INSERT INTO siPartTable PARTITION(country='US') SELECT id,name from dataTab WHERE country='US';
```

#Scenario - We have one file and stores data. For example one file contains all country(IND, UK,US) data. Find a way to filter 'US' data and write data into Partition table with Partition 'US'.



1. Create normal Table and load all data from file -
   - create table dataTab(id int, name string, countryCode string, country string) row format delimited fields terminated by ',' ;
   - load data local inpath '/home/cloudera/partdata/allcountry.csv' into table dataTab;
2. Create Static Partition Table -
   CREATE TABLE siPart(id int, name string, countryCode string) PARTITIONED BY(country string)
   row format delimited fields terminated by ',' location '/user/cloudera/siPartDir';
3. Insert data from normal table in Partition table and create individual Partition
   insert into siPart PARTITION(country=US) SELECT id, name, countryCode from datatab WHERE country='US';
4. Go to HDFS directory and check partition files are created -

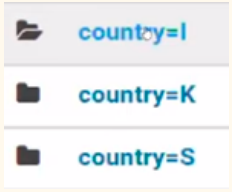
```
[cloudera@quickstart partdata]$ hadoop fs -ls /user/cloudera/slPartDir
Found 3 items
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 00:49 /user/cloudera/slPartDir/country=IND
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 00:54 /user/cloudera/slPartDir/country=UK
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 00:54 /user/cloudera/slPartDir/country=US
```

2. Dynamic Partition -

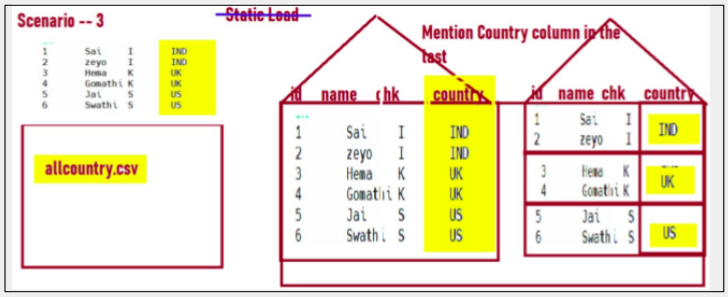
1. CREATE TABLE dynPartTable(id int, name string) PARTITIONED BY(country string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
2. INSERT INTO dynPartTable PARTITION(country) SELECT id, name, country from dataTab;

Rules of Dynamic Partition -

1. The last column will be considered for creating dynamic partition files.
   Like below, provided partition 'country' but the last column was countryCode -



#Scenario - We have one file and stores data. For example one file contains all country(IND, UK,US) data. Find a way to filter 'US' data and write data into dynamic Partition table considering last column 'country' as reference.



1. Create normal Table and load all data from file -
   - create table dataTab(id int, name string, countryCode string, country string) row format delimited fields terminated by ',' ;
   - load data local inpath '/home/cloudera/partdata/allcountry.csv' into table dataTab;
2. Create Dynamic Partition Table taking last column as reference -
   CREATE TABLE dynaPart(id int, name string, countryCode string) PARTITIONED BY(country string)
   row format delimited fields terminated by ',' location '/user/cloudera/dynamicPartDir';
3. Run command to enable dynamic patrition -
   set hive.exec.dynamic.partition.mode=nonstrict;
4. Insert data from normal table in Partition table and create individual Partition
   INSERT INTO dynamicPart PARTITION(country) SELECT id,name,countryCode,country FROM datatab;
5. Go to HDFS directory and check all dynamic partition files are created as per country name -

```
[cloudera@quickstart partdata]$ hadoop fs -ls /user/cloudera/dynamicPartDir
Found 3 items
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 03:13 /user/cloudera/dynamicPartDir/country=IND
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 03:13 /user/cloudera/dynamicPartDir/country=UK
drwxr-xr-x   - cloudera cloudera          0 2022-07-28 03:13 /user/cloudera/dynamicPartDir/country=US
```

Bucketing

Sai Video - https://www.youtube.com/watch?v=WGATmuJTKGQ

- ➢ Used for high cardinal columns for query performance
- ➢ We can use bucking along with Partition
- ➢ Buckets also helps for join during Bucket to Bucket join
- ➢ Good for Integer columns
- ➢ Reduces runs in this operations
- ➢ Bucketing is not folder creation, it's file creation.
- ➢ It use existing column to create buckets

```
1. CREATE EXTERNAL TABLE part_Buck(no INT, name STRING, city STRING, state STRING)
   PARTITIONED BY(country STRING) CLUSTERED BY(no) into 5 BUCKETS
   row format delimited fields terminated by ',' location '/user/cloudera/partBucketData';

2. / * Enable Partition and Bucketing property in HIve - */
```

```
set hive.enforce.bucketing=true;
set hive.exec.dynamic.partition.mode=nonstrict;

3. / * Insert data from Staging table to Partition Bucket target Table */
INSERT INTO part_buck PARTITION (country) SELECT no, name, city, state, country from
stagingtab;
```

Steps -

1. Load data into MySql
   load into infile '/user/cloudera/datafile/data.csv' into table data_tab;

2. Sqoop Import to HDFS from SQL -
   sqoop import --connect jdbc:mysql://localhost/sqldb --username root --password cloudera --table bucktab --m 1 --target-dir /user/cloudera/stagingData

3. Create Staging Hive Table and location HDFS dir-
   CREATE TABLE stagingtab(no INT, name STRING, city STRING, state STRING, country string) row format delimited fields terminated by ',' location '/user/cloudera/stagingData';

4. Create Partition Bucket target Table in Hive
   CREATE EXTERNAL TABLE part_Buck(no INT, name STRING, city STRING, state STRING)
   PARTITIONED BY(country STRING) CLUSTERED BY(no) into 3 BUCKETS
   row format delimited fields terminated by ',' location '/user/cloudera/partBucketData';

5. Enable Partition and Bucketing property in HIve -
   set hive.enforce.bucketing=true;
   set hive.exec.dynamic.partition.mode=nonstrict;

6. Insert data from Staging table to Partition Bucket target Table
   INSERT INTO part_buck PARTITION(country) SELECT no, name, city, state, country from stagingtab;

   Partition and bucketing is created like below for 'country' Partition -

```
Time taken for adding to write entity : 1
Partition default.part_buck{country=INDIA} stats: [numFiles=3, numRows=22, totalSize=241, rawDataSize=460]
Partition default.part_buck{country=India} stats: [numFiles=3, numRows=16, totalSize=159, rawDataSize=302]
Partition default.part_buck{country=U.K} stats: [numFiles=3, numRows=6, totalSize=62, rawDataSize=118]
Partition default.part_buck{country=U.S} stats: [numFiles=3, numRows=6, totalSize=63, rawDataSize=120]
```

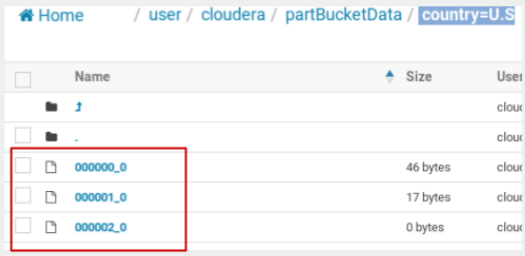
7. Now Go to HDFS location and check Partition Folder and Bucketing files are created -

```
[cloudera@quickstart partdata]$ hadoop fs -ls /user/cloudera/partBucketData/country=India
Found 3 items
-rwxr-xr-x   1 cloudera cloudera         43 2022-07-29 03:32 /user/cloudera/partBucketData/country=India/000000_0
-rwxr-xr-x   1 cloudera cloudera         58 2022-07-29 03:32 /user/cloudera/partBucketData/country=India/000001_0
-rwxr-xr-x   1 cloudera cloudera         58 2022-07-29 03:32 /user/cloudera/partBucketData/country=India/000002_0
[cloudera@quickstart partdata]$ hadoop fs -cat  /user/cloudera/partBucketData/country=India/000000_0
6,Rubal,Kanpur,U.P
3,Ayushi,banglore,DELHI
[cloudera@quickstart partdata]$ hadoop fs -cat  /user/cloudera/partBucketData/country=India/000001_0
7,Rubal,banglore,T.M
4,Adity,LKO,DELHI
1,sai,banglore,T.N
```

How Bucketing decide which part file records will go

Bucketing use Hash Function -

Suppose we have 3 bucketing, now there will be 3 part files created



- Now suppose you have created bucketing for column **id** which contains values like 1,2,3,4,5,6,.....
- **Hash function ⇒**

  Column value % Number of Bucketing = 00000(remainder)_0
- For example id value is = 1

  1 % 3 = 1 (means it will go to **000001_0**)
- For example id value is = 2

  2 % 3 = 1 (means it will go to **000002_0**)
- For example id value is = 3

  3 % 3 = 0 (means it will go to **000000_0**)
- For example id value is = 5

  5 % 3 = 2 (means it will go to **000002_0**)

How to Decide [Bucket Count] in Hive

There is no formula to decide how many buckets should be created but we can trial and error.

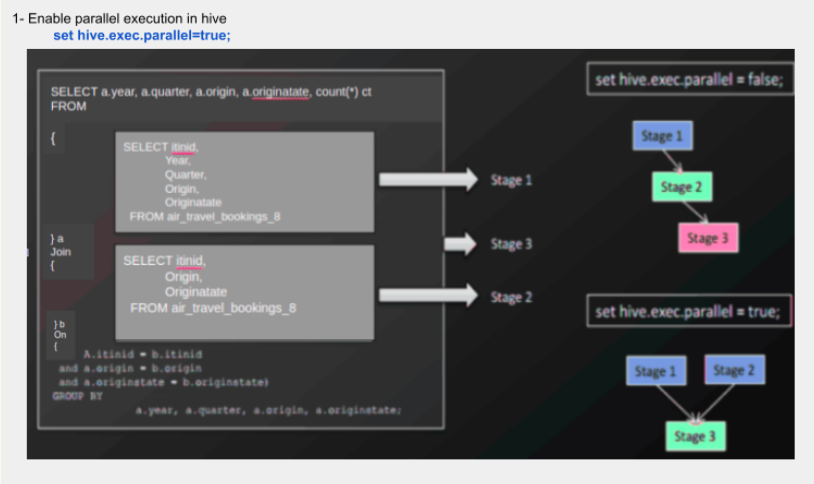
Partition Vs Bucketing

Partition	**Bucketing**
We decide which data have to go in which partition/folder	Here bucketing internally decides using a hash function which record will go in which file.
Partition creates folder and inside that folder we have part-m-00000 file	Bucketing creates part-m-00000 files as per number of buckets
When to use Partitioning?	**When to use Bucketing?**

• When the column with a high search query has low cardinality. For example, if you create a partition by the country name then a maximum of 195 partitions will be made and these number of directories are manageable by the hive. • On the other hand, do not create partitions on the columns with very high cardinality. For example- product IDs, timestamp, and price because it will create millions of directories which will be impossible for the hive to manage. • It is effective when the data volume in each partition is not very high. For example, if you have the airline data and you want to calculate the total number of flights in a day. In that case, the result will take more time to calculate over the partition “Dubai” as it has one of the busiest airports in the world whereas for a country like “Albania” will return results quicker.	• We cannot do partitioning on a column with very high cardinality. Too many partitions will result in multiple Hadoop files which will increase the load on the same node as it has to carry the metadata of each of the partitions. • If some map-side joins are involved in your queries, then bucketed tables are a good option. Map side join is a process where two tables are joined using the map function only without any reduced function. I would recommend you to go through this article for more understanding about map-side joins. • It provides faster query responses like partitioning. • In bucketing due to equal volumes of data in each partition, joins at Map side will be quicker.

Hive Performance Tuning

1. Partitioning
2. Bucketing
3. Parquet
4. Parallel Execution



5. Vectorization

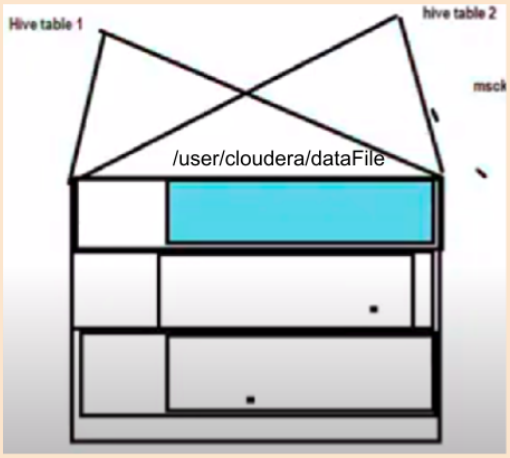
```
set hive.vectorization.execution.enable=true;
```

Usually hive considers only 1 row at a time to process but if you enable vectorization it would consider 1024 rows at a time to process but it kills a lot of memory and core occupancy.

6. Map side join

Task - What will happen if create another partition Bucket Table on the Top of same Location where external Table already created -

Table will be created and it location is same as first one Partition table , Also location have data **but it will show nothing while Query→**



Run below command to Repair this -

- **msck repair table part_buck2;**

```
hive> msck repair table part_buck2;
OK
Partitions not in metastore:    part_buck2:country=INDIA
buck2:country=U.S
Repair: Added partition to metastore part_buck2:country=INDIA
Repair: Added partition to metastore part_buck2:country=India
Repair: Added partition to metastore part_buck2:country=U.K
Repair: Added partition to metastore part_buck2:country=U.S
```

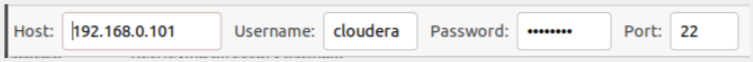
Phase 1 - Project

Phase 1 - Project

Step 1 ⇒ Download winscp (Windows) or Filezilla for (MAC)/Ubuntu

- Windows Download Winscp and install
  https://winscp.net/download/WinSCP-5.19.6-Setup.exe

Step 2 ⇒ Start filezilla (Login Just like putty) and copy prodata to the /home/cloudera/ ⇒



Step 3 ⇒ Cloudera Folks Goto Mysql and create a table

- create table customer_total(id int(10),username varchar(100),sub_port varchar(100),host varchar(100),date_time varchar(100),hit_count_val_1 varchar(100),hit_count_val_2 varchar(100),hit_count_val_3 varchar(100),timezone varchar(100),method varchar(100),`procedure` varchar(100),value varchar(100),sub_product varchar(100),web_info varchar(100),status_code varchar(100));
- load data infile '/home/cloudera/prodata.txt' into table customer_total fields terminated by ',';
- select * from customer_total;
- create table customer_src(id int(10),username varchar(100),sub_port varchar(100),host varchar(100),date_time varchar(100),hit_count_val_1 varchar(100),hit_count_val_2 varchar(100),hit_count_val_3 varchar(100),timezone varchar(100),method varchar(100),`procedure` varchar(100),value varchar(100),sub_product varchar(100),web_info varchar(100),status_code varchar(100));
- insert into customer_src select * From customer_total where id>0 and id<101;

==========
Edge Node
==========

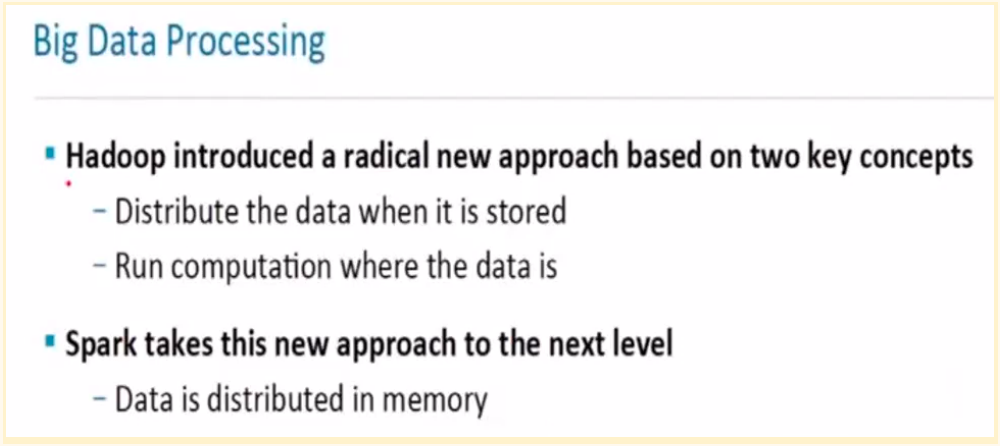
- mkdir /home/cloudera/avsrcdir
- cd /home/cloudera/avsrcdir
- echo -n cloudera>/home/cloudera/passfile
- sqoop job --delete inpjob
- sqoop job --create inpjob -- import --connect jdbc:mysql://localhost/prodb --username root --password-file file:///home/cloudera/passfile -m 1 --table customer_src --target-dir /user/cloudera/customer_stage_loc --incremental append --check-column id --last-value 0 --as-avrodatafile
- sqoop job --exec inpjob
- hadoop fs -mkdir /user/cloudera/avscdirpro
- hadoop fs -put /home/cloudera/avsrcdir/customer_src.avsc /user/cloudera/avscdirpro

==========
Hive shell
==========

- create table customer_src ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED AS AVRO LOCATION '/user/cloudera/customer_stage_loc' TBLPROPERTIES ('avro.schema.url'='/user/cloudera/avscdirpro/customer_src.avsc');
- select * from customer_src; === U will see the data
- create external table customer_target_tab partitioned by (current_day string,year string,month string,day string) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' STORED AS AVRO LOCATION '/user/cloudera/customer_target_tab' TBLPROPERTIES ('avro.schema.url'='/user/cloudera/avscdirpro/customer_src.avsc');
- select * from customer_target_tab; ==== U will not see the data

SPARK

was Invented by Matai Zaharia in 2010 but was initially released by Apache in 2014.



Spark Installation

1. Prerequisites ⇒ Java and Scala
2. Download Java in case it is not installed using the below commands.

```
sudo apt-get install python-software-properties
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

3. Download the latest Scala version from Scala Lang Official page. Once installed, set the scala path in the **~/.bashrc** file as shown below.

```
export SCALA_HOME=Path_Where_Scala_File_Is_Located
export PATH=$SCALA_HOME/bin:PATH
```

4. Download Spark from the Apache Spark Downloads page.
5. Extract Spark tar using the below command.

```
tar -xvf spark-2.1.0-bin-hadoop2.7.tgz
```

6. Set the Spark_Path in **~/.bashrc** file.

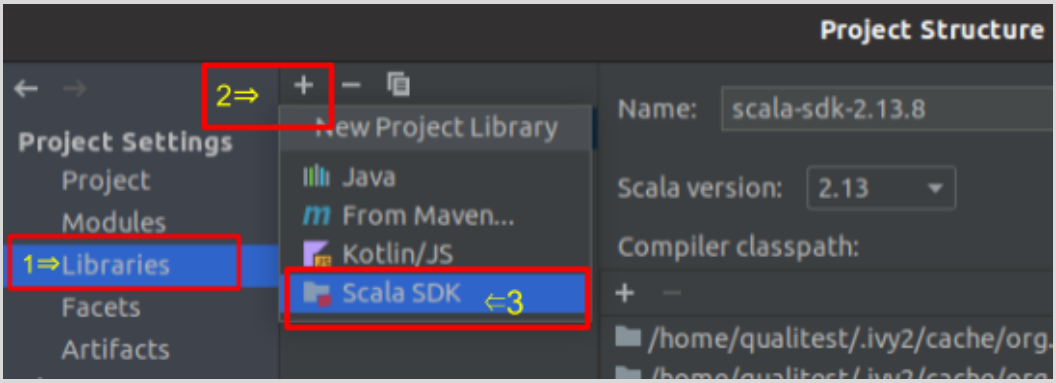
```
export SPARK_HOME=Path_Where_Spark_Is_Installed
export PATH=$PATH:$SPARK_HOME/bin
```

Spark set up with Scala and Run on Intellij

1. Make sure Java is installed on 8/11/17. (windows - Download winUtils and save in dir → **D:/hadoop/bin/winutils.exe** and set environment variable path)

i. HADOOP_HOME ⇒ **D:/hadoop**
ii. Update **'Path'** variable with add a new path item there ⇒ "**%HADOOP_HOME%\bin**"

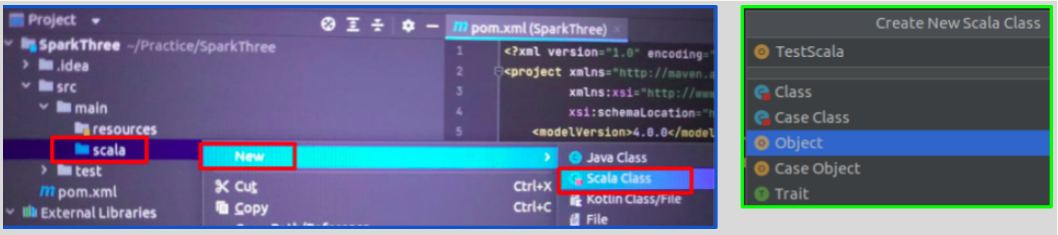
2. Create Maven Project with Java
3. Install Scala Plugin in Intellij
4. Rename **/src/main/java** with **/src/main/scala**
5. **Setup Scala SDK** - Go to **Project-structure in IntelliJ** → Go to Libraries → **Add Scala SDK** → Click on APPLY and click OK



6. **Add Spark Dependencies** → Now either [Add Spark Jar files](spark-2.4.7-bin-hadoop2.7.tgz) (spark-2.4.7-bin-hadoop2.7.tgz → jars folder) or **Add Spark Maven dependency** in pom.xml

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId> spark-core_2.12</artifactId>
  <version>3.3.0</version>
</dependency>

 <dependency>
   <groupId>org.apache.spark</groupId>
   <artifactId>spark-sql_2.12</artifactId>
   <version>3.3.0</version>
   <scope>compile</scope>
</dependency>
```

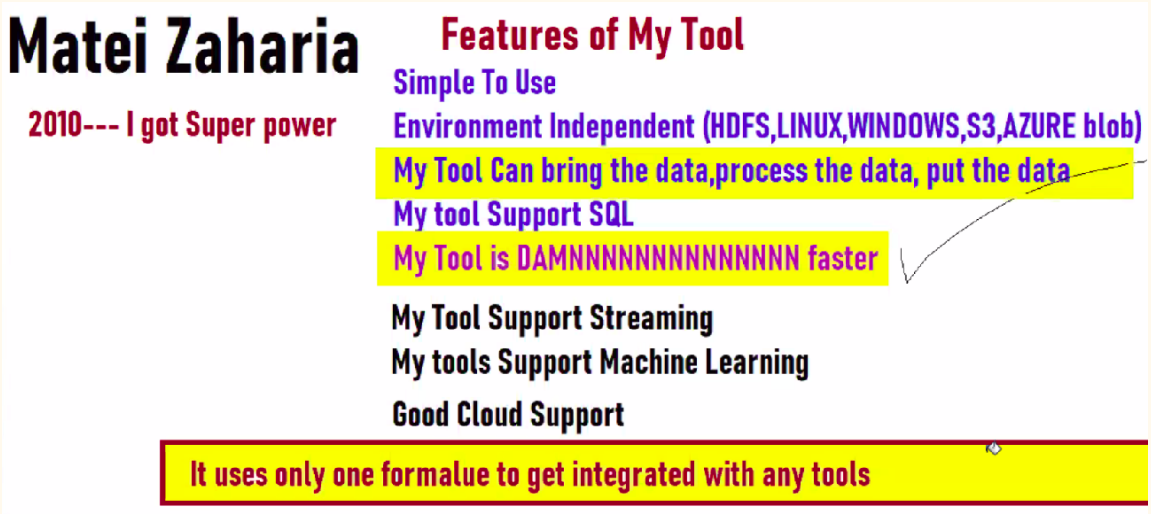
7. Create first Scala Object under **/src/main/scala**
   i. Right Click on /**scala**/ package and select **New**
   ii. Select Scala class
   iii. Give the object name and select **scala object.** Like the below image -



8. Run Maven Build → `mvn clean install`
9. Now run the Spark application **SparkSessionTest** program.
10. Run jar in Cloudera - **spark-submit --class pack.test SparkThree-1.0-SNAPSHOT.jar**

Why does Spark come into the Market?

1. Hive used to satisfy only a few requirements.
2. Hive runs on MR
3. Hive is quite slow for very very very large data set
4. Hive RUNS on HDFS (No other platform)
5. Hive No support on Streaming
6. Hive has no support for Machine Learning
7. Hive cannot integrate with any sources to bring the data
8. Hive is not customizable easily



Advantages of Spark

- faster
- in-memory processing
- lazy computations
- environment independent
- MLV streaming
- Cloud Support
- Customization Serialization
- Unified Formulae (easy to read and write data from sources)
- GraphDB support
- SQL

Why is SPARK Faster?

- Spark uses in-memory processing while MR(MapReduce uses I/O processing)
- Lazy in nature - Once Action is triggered then only Transactions will run
    - Transaction 1 <=== Transaction 2 <=== Action
- The lightning-fast performance due to its In-Memory Processing Capability brought Spark its place amongst the top-level Apache Projects.

Brief Introduction to Apache Spark

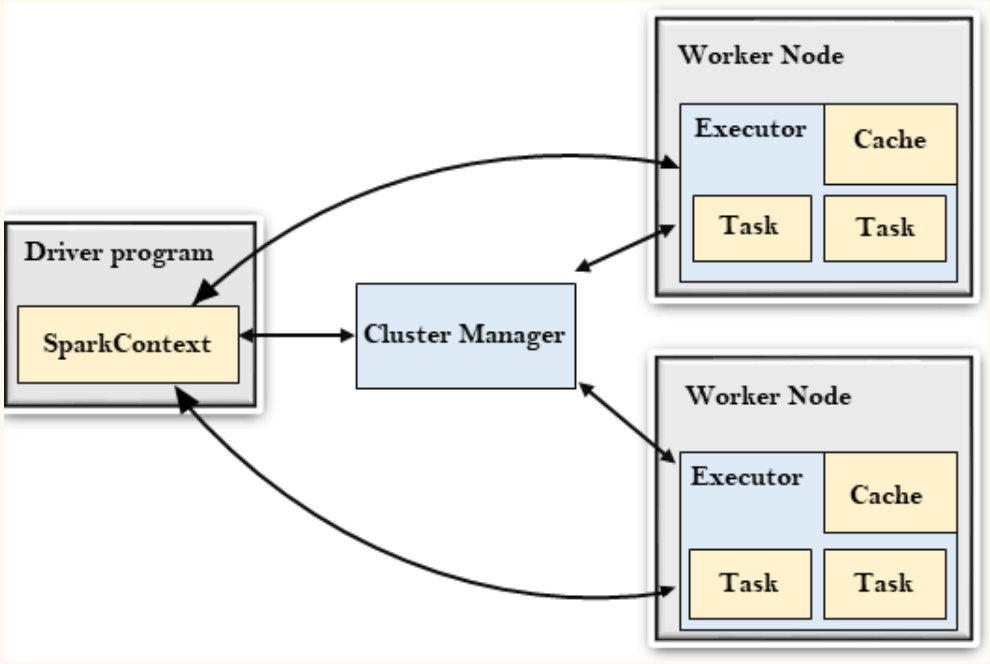
- ➢ Apache SparkTM is a fast and general engine for large-scale data processing framework for massive parallel computing (cluster)
- ➢ Harnessing the power of cheap memory
- ➢ Written using Scala, Java, and Python languages
- ➢ High-level APIs support in Scala, Java and Python
- ➢ Has had 36,338 commits made by 1,319 contributors. Representing lakhs of lines of code with proper comments.
- ➢ Developers from 50+ companies including UC Berkeley, Cloudera, Yahoo, Databricks, Intel, Groupon, etc., Apache Committers from 16+ organizations

Spark Architecture

Spark's Basic Architecture

Typically when you think of a "computer" you think about one machine sitting on your desk at home or at work. This machine works perfectly well for watching movies or working with spreadsheet software. However, as many users likely experience at some point, there are some things that your computer is not powerful enough to perform. One particularly challenging area is data processing. Single machines do not have enough power and resources to perform computations on huge amounts of information (or the user may not have time to wait for the computation to finish). A *cluster*, or group of machines, pools the resources of many machines together allowing us to use all the cumulative resources as if they were one. Now a group of machines alone is not powerful, you need a framework to coordinate work across them. Spark is a tool for just that, managing and coordinating the execution of tasks on data across a cluster of computers.

The cluster of machines that Spark will leverage to execute tasks will be managed by a cluster manager like Spark's Standalone cluster manager, YARN, or Mesos. We then submit Spark Applications to these cluster managers which will grant resources to our application so that we can complete our work.



When Spark submit is done, Spark context gets created in which it runs Driver program which submits the Job to the cluster manager which inturn Communicates with each worker node.

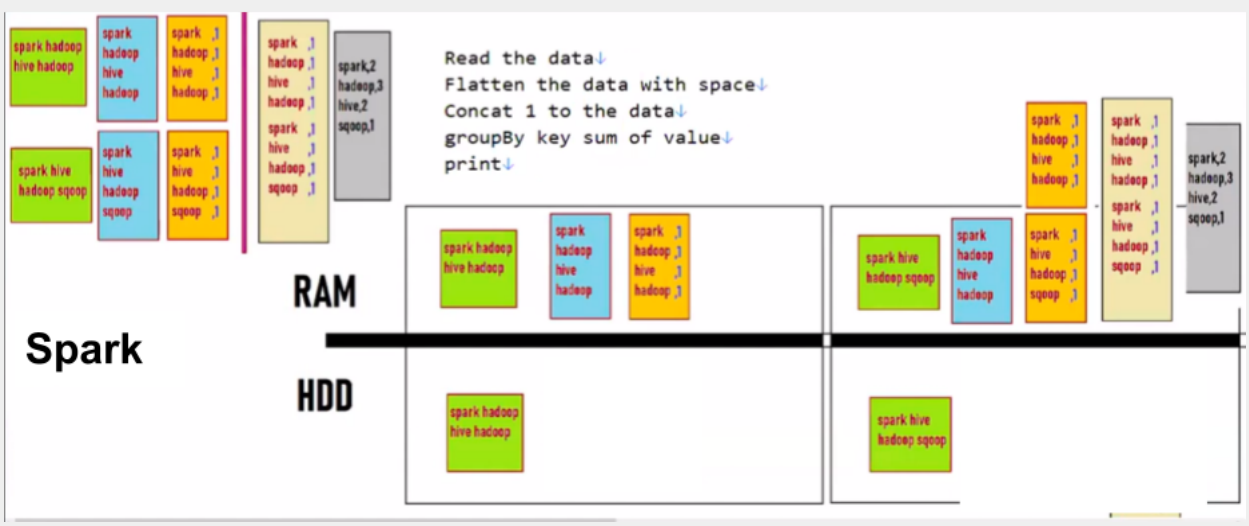
SPARK Vs MapReduce

Spark	MR

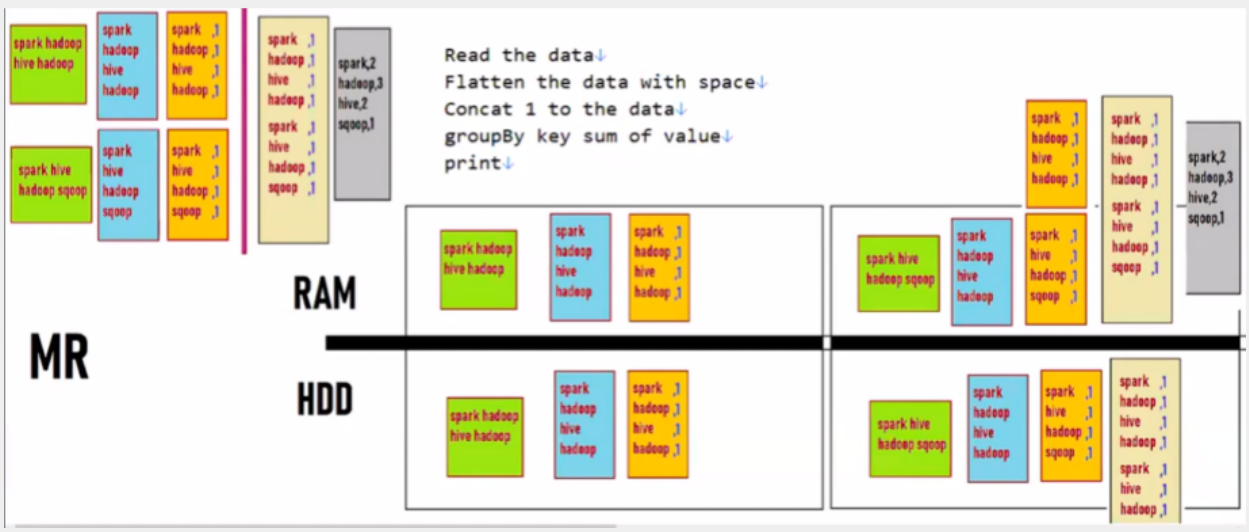
<table>
<tr><td>In-memory processing</td><td>I/O processing</td></tr>
<tr><td>Lazy evolution</td><td>Linear - Run steps one by one</td></tr>
<tr><td>Till the Action is triggered none of the Transaction will work.

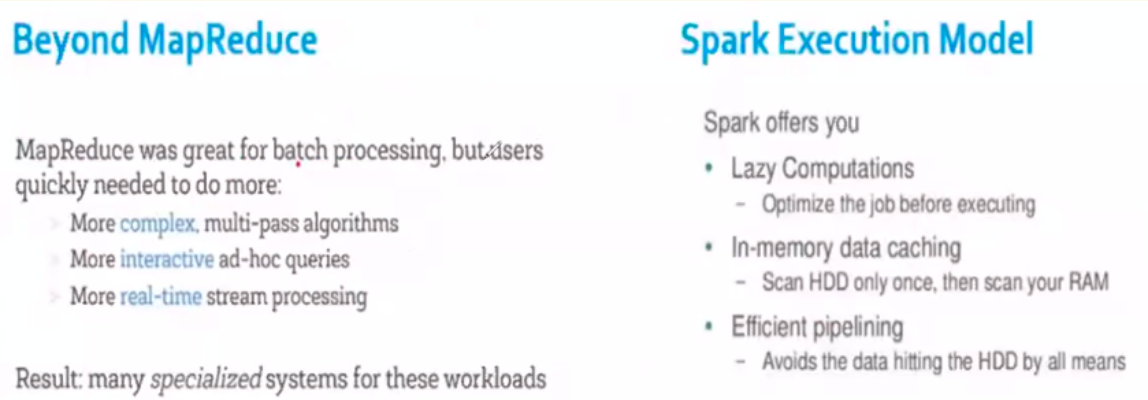
<table><tr><th>Transactions</th><th>Actions</th></tr><tr><td>Join
Filter, map</td><td>Count
Select</td></tr></table></td><td>Operation run step by step</td></tr>
</table>

Spark - In-Memory Processing example -



MR - I/O Processing example -





Pillars of Spark

Direct Acyclic Graph - sequence of computations performed on data

- ➢ Node - RDD partition
- ➢ Edge - transformation on top of data
- ➢ Acyclic - graph cannot return to the older partition
- ➢ Direct-transformation is an action that transitions data partition state (suppose from A node to B node)

RDD

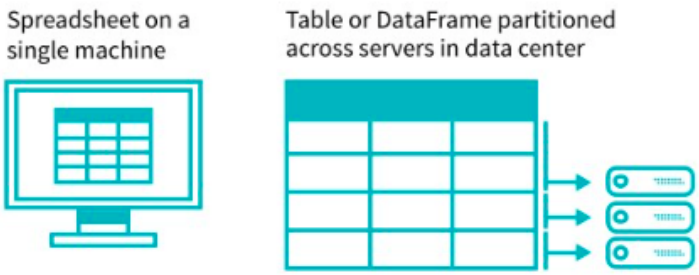
1. RDD means -
   - ★ **Resilient** - if data in memory is lost, it can be recreated
   - ★ **Distributed** - stored in memory across the cluster
   - ★ **Dataset** - initial data can come from a file or be created programmatically
2. RDDs are the fundamental unit of data in Spark
3. Most Spark programming consists of performing operations on RDDs
4. It is fault-tolerant if you perform multiple transformations on the RDD and then any node fails for any reason. The RDD, in that case, is capable of recovering automatically.
5. it does not have an optimizer like catalyst
6. it is recommended to use a Data frame because of its performance
7. it is available from 1.0 version
8. ## RDD Advantages
   - In-Memory Processing
   - Immutability
   - Fault Tolerance

- Lazy Evolution
- Partitioning
- Parallelize

Dataframe

★ It is conceptually a Row Columnar format
★ It has a optimizer like catalyst for optimization, they are capable enough to process large datasets.
★ It is available from the 1.3 version
★ Capable enough to handle structured data
★ Support for various data formats, such as Hive, CSV, XML, JSON, RDDs, Cassandra, Parquet, etc

A *DataFrame* is the most common Structured API and simply represents a table of data with rows and columns. The list of columns and the types in those columns the *schema*. A simple analogy would be a spreadsheet with named columns. The fundamental difference is that while a spreadsheet sits on one computer in one specific location, a Spark DataFrame can span thousands of computers. The reason for putting the data on more than one computer should be intuitive: either the data is too large to fit on one machine or it would simply take too long to perform that computation on one machine.



Datasets

★ Spark Datasets is an extension of Dataframes API with the benefits of both RDDs and the Datasets.
★ It is fast as well as provides a type-safe interface. Type safety means that the compiler will validate the data types of all the columns in the dataset while compiling only and will throw an error if there is any mismatch in the data types.

The next topic we'll cover is a type-safe version of Spark's structured API for Java and Scala, called *Datasets*. This API is not available in Python and R, because those are dynamically typed languages, but it is a powerful tool for writing large applications in Scala and Java.

Recall that DataFrames, which we saw earlier, are a distributed collection of objects of type Row, which can hold various types of tabular data. The Dataset API allows users to assign a Java class to the records inside a DataFrame, and manipulate it as a collection of typed objects, similar to a Java `ArrayList` or Scala `Seq`. The APIs available on Datasets are *type-safe*, meaning that you cannot accidentally view the objects in a Dataset as being of another class than the class you put in initially. This makes Datasets especially attractive for writing large applications where multiple software engineers must interact through well-defined interfaces.

The Dataset class is parametrized with the type of object contained inside: `Dataset<T>` in Java and `Dataset[T]` in Scala. As of Spark 2.0, the types `T` supported are all classes following the JavaBean pattern in Java, and `case classes` in Scala. These types are restricted because Spark needs to be able to automatically analyze the type `T` and create an appropriate schema for the tabular data inside your Dataset.

The awesome thing about Datasets is that we can use them only when we need or want to. For instance, in the follow example I'll define my own object and manipulate it via arbitrary map and filter functions. Once we've performed our manipulations, Spark can automatically turn it back into a DataFrame and we can manipulate it further using the hundreds of functions that Spark includes. This makes it easy to drop down to lower level, perform type-safe coding when necessary, and move higher up to SQL for more rapid analysis. We cover this material extensively in the next part of this book, but here is a small example showing how we can use both type-safe functions and DataFrame-like SQL expressions to quickly write business logic.

Spark processes the data in Two Ways

★ RDD and the other one is Dataframe

RDD

1. **For the RDD process, we need a jar "Spark-core.jar" which contains the class name "SparkContext" which has many methods like textFile(), filter, map(), flatMap(), etc.**
2. **To import 'SparkContext' in the scala file write →`import org.apache.spark.SparkContext`**

Below is the example of the RDD query and output -

1. —---- open Spark shell —---
   spark-shell

2. sc.textFile("file:///home/cloudera/data.csv").filter(x => x.contains("US")).foreach(println)

```
5,Jai,S,US
6,Swathi,S,US
```

```
======== How to Create SparkContext Object ==============

import org.apache.spark.SparkContext

object SparkWordCount {
  def main(args: Array[String]) {

    /* "setAppName()" –> This is the name of the application that you want to run.
       "setMaster()"  --> This parameter denotes the master URL to connect the spark application to.  */
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")

        /* Creating a [RDD] SparkContext Object */
        val sc = new SparkContext(conf)

        /* Read input file and store in variable */
        val input = sc.textFile("input.txt")
        val count = input.flatMap(line ⇒ line.split(" "))
                         .map(word ⇒ (word, 1))
                         .reduceByKey(_ + _)
         count.saveAsTextFile("outfile")
          }
        }
```

- Use local[x] when running in Standalone mode. x should be an integer value greater than 0; this represents how many partitions it should create when using RDD, DataFrame, and Dataset. Ideally, the x value should be the number of CPU cores you have.

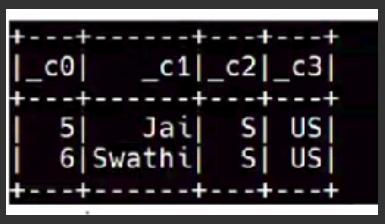
DataFrame → SPARK SQL

★ **For the DataFrame process, we need a jar "Spark-sql.jar" which contains the class name "SparkSession" which has many methods like read(), filter, etc.**

★ **To import SparkSession in scala file write →`import org.apache.spark.sql.SparkSession`**

Below is the example of the RDD query and output -

1. **spark-shell**
2. spark.read.csv("file:///home/cloudera/data.csv").filter("_C3='US'").show()



```
+---+------+---+---+
|_c0|   _c1|_c2|_c3|
+---+------+---+---+
|  5|   Jai|  S| US|
|  6|Swathi|  S| US|
+---+------+---+---+
```

======== How to Create SparkContext Object ==========

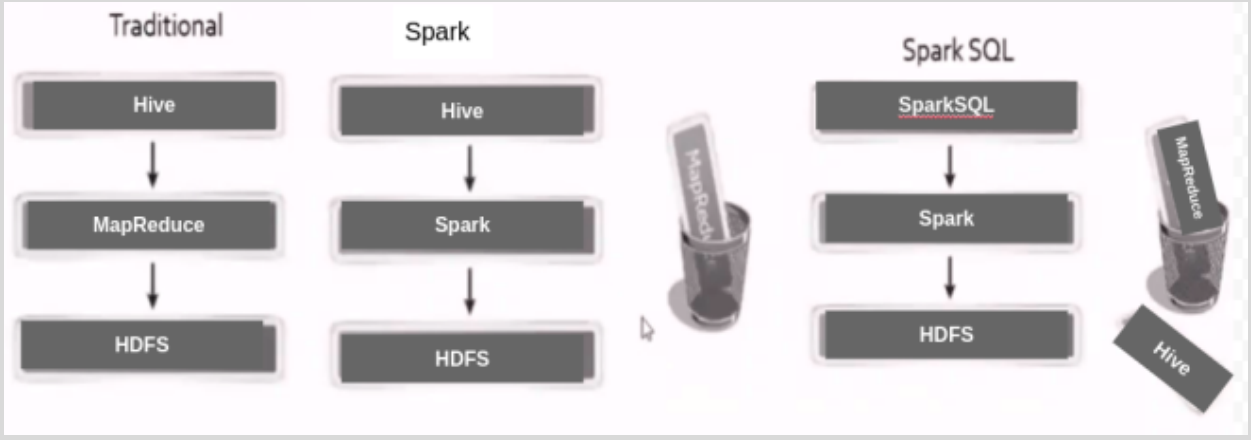
```
import org.apache.spark.sql.SparkSession

object SparkWordCount {
  def main(args: Array[String]) {

  /*  SparkSession.builder() – Return SparkSession.Builder class. This is a builder for SparkSession.
       master(), appName() and getOrCreate() are methods of SparkSession.Builder.

      “appName()” –> This is the name of the application that you want to run.
      “master()”  --> This parameter denotes the master URL to connect the spark application to.
   /
    Creating a [DataFrame] Spark Session Object */
          var sparkSession = SparkSession.builder()
                             .master("local[1]")
                            .appName("SparkByExamples.com")
                           .getOrCreate();

          /* Read input file and store in variable */
          val input = sc.read.csv("input.csv")
            }
          }
```

RDD Vs DataFrame

RDD	DataFrame
Low Level —> If the requirement is small and easy, then RDD is a good choice.	**High Level —>** If the requirement is huge and big scenarios, then DF is a good choice. Like have semi-structured data
Support only **Structured** data	Support **Structure and semi-structured**

Through RDD, we can process structured as well as unstructured data. But, in RDD user need to specify the schema of ingested data, RDD cannot infer its own	In the data frame data is organized into named columns. Through dataframe, we can process structured and unstructured data efficiently. It also allows Spark to manage schemas.
Performance is **slow** as compared DF	Performance is faster as compared to RDD. DF has an inbuilt **Tungston framework** which makes DF to perform faster.
Less integration to read data from external source - It can only ready files, There are some extend to read data from Hbase, Hive, Cassandra, NoSQL but it is very very difficult	**Very good integration to read data from external source due to Unified formula** (`spark.read.format("csv").load("path")`)

Evolution of Spark SQL



Transformations (Narrow and Wide)

In Spark, the core data structures are *immutable* meaning they cannot be changed once created. This might seem like a strange concept at first, if you cannot change it, how are you supposed to use it? In order to "change" a DataFrame you will have to instruct Spark how you would like to modify the DataFrame you have into the one that you want. These instructions are called *transformations*. Let's perform a simple transformation to find all even numbers in our currentDataFrame.

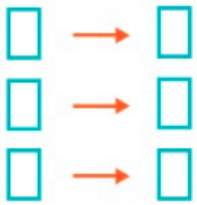
```
%scala

val divisBy2 = myRange.where("number % 2 = 0")
```

```
%python

divisBy2 = myRange.where("number % 2 = 0")
```

You will notice that these return no output, that's because we only specified an abstract transformation and Spark will not act on transformations until we call an action, discussed shortly. Transformations are the core of how you will be expressing your business logic using Spark. There are two types of transformations, those that specify narrow dependencies and those that specify wide dependencies.

Transformations consisting of *narrow dependencies*s (we'll call them narrow transformations) are those where each input partition will contribute to only one output partition. In the preceding code snippet, our `where` statement specifies a narrow dependency, where only one partition contributes to at most one output partition.

Narrow Transformations
1 to 1



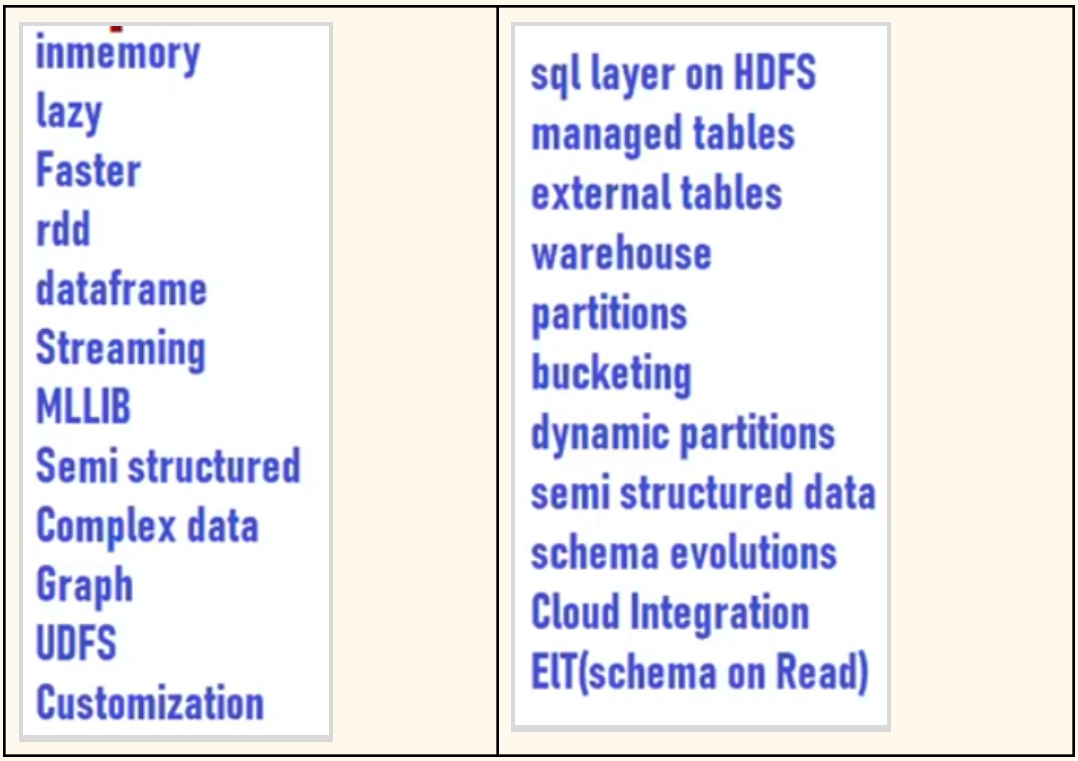
A *wide dependency* (or wide transformation) style transformation will have input partitions contributing to many output partitions. You will often hear this referred to as a *shuffle* where Spark will exchange partitions across the cluster. With narrow transformations, Spark will automatically perform an operation called pipelining on narrow dependencies, this means that if we specify multiple filters on DataFrames they'll all be performed in-memory. The same cannot be said for shuffles. When we perform a shuffle, Spark will write the results to disk. You'll see lots of talks about shuffle optimization across the web because it's an important topic but for now all you need to understand are that there are two kinds of transformations.

Wide Transformations (shuffles)
1 to 1

We now see how transformations are simply ways of specifying different series of data manipulation. This leads us to a topic called lazy evaluation.

Hive Integration with Spark

Spark Features	Hive features



For hive integration with spark, both tools should have been installed in the same cluster.

How Hive and Spark integrated

Hive has the file name "**hive-site.xml**", we have to copy this file in spark configuration path as well.

To check hive-site.xml file in hive, check location -

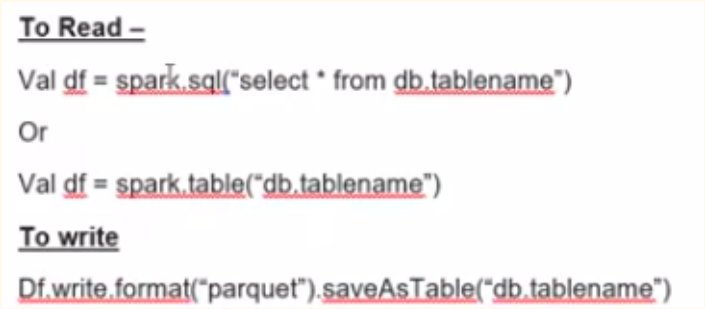
```
ls /etc/hive/conf
```

To check hive-site.xml file in spark, check hive location -

```
ls /usr/local/spark/conf
```

Hive works with metastore, so to work with spark, spark also listen same metastore

Read/Write from Hive



➔ **If we don't provide format while writing data to Hive, data will be save as Parquet, (when we don't provide any location data will be created in hive warehouse)**

```
df.write.saveAsTable("db.test")
```

➔ If table is already created in Hive and just need to insert data in table then just give format 'hive', which means hive will store data in same format, table is created. suppose table is created for ORC format then using below command data will store in ORC →

```
df.write.format("hive").mode("append").saveAsTable("db.test")
```

Scala programs

Practice more programs here - https://www.w3resource.com/scala-exercises/basic/index.php

Word Count Program

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._

object SparkWordCount {
  def main(args: Array[String]) {

    /* "setAppName()" –> This is the name of the application that you want to run.
       "setMaster()"  --> This parameter denotes the master URL to connect the spark application to. */
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")

        /* Creating a [RDD] SparkContext Object */
        val sc = new SparkContext(conf)
```

```
/* Read input file and store in variable */
val input = sc.textFile("input.txt")
val count = input.flatMap(line ⇒ line.split(" "))
                .map(word ⇒ (word, 1))
                .reduceByKey(_ + _)
 count.saveAsTextFile("outfile")
  }
}
```

Program - How to use Filter, Map, FaltMap. Below is the List ⇒

"Amazon-Jeff-America",
'Microsoft-BillGates-America",
'TCS-TATA-india",
'Reliance-Ambani-INDIA"

Below are the operation to do on the above list ⇒

- Filter 'india' records (Ensure capital INDIA also filtered)
- Flatten(split) the result with Hyphen
- Then replace "india" with "local" value
- Then Concatenate "- Done" at the end of each item of the list then print it

```
object obj {

 def main(args: Array[String]): Unit = {
  var liststr = List("Amazon-Jeff-America",
    "Microsoft-BillGates-America",
    "TCS-TATA-india",
    "Reliance-Ambani-INDIA")

  println("String List ==> " + liststr)

  var list2 = liststr.filter(x => x.toLowerCase().contains("india"))
    .flatMap(x => x.split("-"))
    .map(x => x.replace("india", "local"))
    .map(x => x.concat(", Done"))

  println("filter list ==> ")
  list2.foreach(println)

===== OUTPUT ======
TCS- Done
TATA- Done
local- Done
```

```
Reliance- Done
Ambani- Done
INDIA- Done
```

 # Write a Scala program to compute the sum of the two given integer values. If the two values are the same, then return triples of their sum.

```scala
package pack

object obj2 {

 def main(args: Array[String]): Unit={
   print(sum(1,2))
   print(sum(2,2))
   print(sum(4,4))
 }

 def sum(i: Int, i1: Int): Int={
   if (i == i1) (i + i1) * 3 else (i + i1)
 }
}
```

RDD - Use Case #1

- Read the data file (data.csv) [just copy below data in a file]
- /*
  first_name,last_name,company_name,address,city,county,state,zip,age,phone1,phone2,email,web
  James,Butt,"Benton, John B Jr",6649 N Blue Gum St,New
  Orleans,Orleans,LA,70116,9,504-621-8927,504-845-1427,jbutt@gmail.com,http://www.bentonjohnbjr.com
  Josephine,Darakjy,"Chanay, Jeffrey A Esq",4 B Blue Ridge
  Blvd,Brighton,Livingston,MI,48116,8,810-292-9388,810-374-9840,josephine_darakjy@darakjy.org,http://www.chan
  ayjeffreyaesq.com
  Art,Venere,"Chemel, James L Cpa",8 W Cerritos Ave
  #54,Bridgeport,Gloucester,NJ,8014,7,856-636-8749,856-264-4130,art@venere.org,http://www.chemeljameslcpa.com
  */
- Filter rows greater than which length > 200
- Flatten(split) the Filtered data with comma
- Replace hyphen with Nothing (Remove the Hyphen)
- Concat string →",zeyo" at the End of each element
- Store result in a file

==========

```scala
package pack
import org.apache.spark.{SparkConf, SparkContext}
object obj1 {

 def main(args:Array[String]): Unit={
   val conf = new SparkConf().setAppName("Spark practice").setMaster("local[*]")
   val sc = new SparkContext(conf)

   /* Read the data file */
   var data = sc.textFile("file:///home/Practice/data/data.csv")

   data = data.filter(x => x.length > 200)     /* Filter rows greater than which length > 200 */
     .flatMap(x => x.split(","))         /* Flatten(split) the Filtered data with comma */
     .map(x => x.replace("-", ""))  /* Replace hyphen  with Nothing (Remove the Hyphen) */
     .map(x => x.concat(" -> zeyo done"))   /* Concat string →",zeyo" at the End of each element */

   data.foreach(println)

  /* Data write to a file (provide non-exist folder name in below)*/
  data.coalesce(1).saveAsTextFile("file:///home/Practice/resultData")
 }
}
```

RDD with DF- Use Case #2

1. Read data from text file
2. Split each column with comma like below

```
00000000,06-26-2011,Exercise,GymnasticsPro
00000002,06-01-2011,Exercise,GymnasticsPro
00000003,06-05-2011,Gymnastics,Rings


/* After split using map() */
------------------------------------------------------------
index 0        1              2              3
------------------------------------------------------------
00000000       06-26-2011     Exercise       GymnasticsPro
00000002       06-01-2011     Exercise       GymnasticsPro
00000003       06-05-2011     Gymnastics       Rings
```

3. **Impose(create) schema using case class**
4. Convert it into schemaRDD
5. Filter products(column/index 4) element contains “Gymnastics”
6. Convert it into DataFrame

7. Save result as Parquet

==========

Using SchemaRDD

==========

```
package pack
import org.apache.spark.sql.SparkSession
import org.apache.spark.{SparkConf, SparkContext}
object obj1 {

 case class schema(id: String, date: String, category: String, product: String) /* define schema using case class */

 def main(args: Array[String]): Unit = {
   val conf = new SparkConf().setAppName("spark practice").setMaster("local[*]")
   val sc = new SparkContext(conf)

   val data = sc.textFile("file:///home/Practice/data/datatxns.txt")

   val splitData = data.map(x => x.split(",")) /* split data */

   /* Using SchemaRDD*/
   val rddSchemaResult = splitData.map(x => schema(x(0), x(1), x(2), x(3))) /* convert data into schema(case class) */
     .filter(x => x.product.contains("Gymnastics"))  /* filter data */

   println("====== RDD result for 'Gymnastics' product =======")
   rddSchemaResult.foreach(println)

   println("====== Start DataFrame =======")
   val spark = new SparkSession.Builder().getOrCreate()
   import spark.implicits._

   val df = rddSchemaResult.toDF() /* convert rdd result into dataframe */
   df.createOrReplaceTempView("filterTable") /* create sql view to perform more operation on data*/
   val df_result = spark.sql("SELECT * from filterTable where date='06-26-2011'") /* run sql query on created view */

   println("====== Filter more data using DataFrame =======")
   df_result.show() /* print data */
   df_result.coalesce(1).write.parquet("file:///home/Practice/data/parquetData") /* save data in parquet file*/
 }
}
```

Solution 2 → Run same able scenario but instead of using schemaRDD(case class) use Row rdd

1. **Step (1) & 2 is same as above solution**
2. **After split data now CONVERT it to ROW rdd**

3. Filter products(column/index 4) element contains “Gymnastics”
4. **Impose(create) schema using StructType()**
5. Convert it into DataFrame
6. Save result as Parquet

==========

Using RowRDD

==========

```
package pack
import org.apache.spark.sql.types.{StringType, StructField, StructType}
import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}

object obj1 {
 def main(args: Array[String]): Unit = {
   val conf = new SparkConf().setAppName("spark practice").setMaster("local[*]")
   val sc = new SparkContext(conf)
   val data = sc.textFile("file:///home/Practice/data/datatxns.txt")

   val splitData = data.map(x => x.split(",")) /* split data */

   /* Using Row RDD*/
   val rowData = splitData.map(x => Row(x(0), x(1), x(2), x(3))) /* convert data into Row rdd  */
     .filter(x => x(3).toString.contains("Gymnastics")) /* filter data */

   println("====== RDD result for 'Gymnastics' product =======")

   rowData.foreach(println)

   val structSchema = StructType(Array(
     StructField("id", StringType, true),
     StructField("date", StringType, true),
     StructField("category", StringType, true),
     StructField("product", StringType, true)
   ))

   println("====== Start DataFrame =======")
   val spark = new SparkSession.Builder().getOrCreate()

  val df = spark.createDataFrame(rowData, structSchema) /* create frame using rdd data and struct schema */

   df.createOrReplaceTempView("filterTable")
   /* create sql view to perform more operation on data*/
```

```
   val df_result = spark.sql("SELECT * from filterTable where date='06-26-2011'") /* run sql query on created view */

   println("====== Filter more data using DataFrame =======")
   df_result.show() /* print data */
   df.coalesce(1).write.parquet("file:///home/Practice/data/parquetData") /* save data in parquet file*/
 }
}
```

Ways of Creating RDD

```
# 1. parallelizing data collection
my_list = [1, 2, 3, 4, 5]
my_list_rdd = sc.parallelize(my_list)
my_list_rdd = sc.parallelize(List(1, 2, 3, 4, 5))

## 2. Referencing to external data file
file_rdd = sc.textFile("path_of_file")
```

Ways of Create Dataframe

```
=========================================================
Create DataFrame from RDD
=========================================================
val columns = Seq("language","users_count")
val data = Seq(("Java", "20000"), ("Python", "100000"), ("Scala", "3000"))
val rdd = sc.parallelize(data)
val dfFromRD1 = rdd.toDF(columns)
```

```
=========================================================
Using createDataFrame() with the Row type
=========================================================
val data = Seq(("Java", "20000"), ("Python", "100000"), ("Scala", "3000"))
val rdd = sc.parallelize(data)
val schema = StructType(Array(
                StructField("language", StringType,true),
                StructField("users", StringType,true)
            ))
```

```
val rowRDD = rdd.map(x => Row(x._1, x._2))
val dfFromRDD = spark.createDataFrame(rowRDD,schema)
```

```
=====================================================
Using createDataFrame() from SparkSession
=====================================================
val columns = Seq("language","users_count")
val data = Seq(("Java", "20000"), ("Python", "100000"), ("Scala", "3000"))
val rdd = sc.parallelize(data)
var dfFromData = spark.createDataFrame(data).toDF(columns:_*)
```

Read data from CSV/TEXT/JSON/Parquet/ORC (These file type are supported by Spark by default)

```
##  First Way to read file
val df = spark.read.csv("/src/resources/file.csv")
val df = spark.read.text("/src/resources/file.txt")
val df = spark.read.json("/src/resources/file.json")


## Second Way to read file
val df = spark.read.format("csv").options("header", true).load("file:///C:/data/datatxns.txt")
val df = spark.read.format("json").load("file:///C:/data/devices.json")
val df = spark.read.format("orc").load("file:///C:/data/data.orc")
val df = spark.read.format("parquet").load("file:///C:/data/data.parquet")
```

Read data from Avro

(need to add jar or dependency -> spark-avro_2.12)

```
=============================================================================
val df = spark.read.format("avro").load("file:///home/Practice/data.avro")
```

```
=============================================================================
```

Read data from XML

(need to add jara or dependency -> spark-xml_2.12)

```
=============================================================================
```

```scala
val df = spark.read.format("xml")
     .option("rootTag", "POSLog")
     .option("rowTag", "Transaction")
     .load("file:///home/Practice/data/transactions.xml")
```

------- Below is xml -----------

```
<POSLog>
  <Transaction>
    <RetailStoreID>48</RetailStoreID>
    <WorkstationID>6</WorkstationID>
    <SequenceNumber>73</SequenceNumber>
    <BusinessDayDate>2014-09-30</BusinessDayDate>
    <EndDateTime>2014-09-30T06:20:14</EndDateTime>
    <OperatorID OperatorName="KERRY P">48237</OperatorID>
    <CurrencyCode>USD</CurrencyCode>
    <RetailTransaction Version="2.1">
    ...
    </RetailTransaction>
  </Transaction>
  <Transaction>
  ...
  </Transaction>
  <Transaction>
  ...
  </Transaction>
  <Transaction>
  ...
  </Transaction>
  <Transaction>
  ...
  </Transaction>
</POSLog>
```

Read data from RDBMS Database

(need to add jara or dependency –> `mysql-connector-java`)

==========

```scala
val df_mysql = spark.read.format("jdbc")
   .option("url", "jdbc:mysql://localhost:port/db")
   .option("driver", "com.mysql.jdbc.Driver")
   .option("dbtable", "tablename")
   .option("user", "user")
   .option("password", "password")
   .load()
```

Read data from AWS s3 location file

(need to add jara or dependency –> `aws-java-sdf`)

```
=========================================================
/* Read file from AWS S3 location */
val df = spark.read.format("csv")
 .option("header", true)
 .option("fs.s3a.access.key", "access-key")
 .option("fs.s3a.secret.key", "secret-key")
 .load("s3a://zeyodevbb/datatxns.txt")

df.show()
```

Read data from Hive Table

```
==================================
val df = sparkSession.read
    .options(Map(HBaseTableCatalog.tableCatalog -> catalog))
    .format("org.apache.spark.sql.execution.datasources.hbase")
    .load()
```

Read data from Snowflake

database is a purely cloud-based data storage and analytics Dataware house provided as a Software-as-a-Service (SaaS).
(need to add jar or dependency –> `spark-snowflake_2.12`)

```
====================================
val snowdf = spark.read.format("snowflake")
                              .option("sfURL","https://YI78860.ap-southeast-1.snowflakecomputing.com")
                              .option("sfAccount","YI78860")
                              .option("sfUser","zeyoanalytics2")
                              .option("sfPassword","Aditya908")
                              .option("sfDatabase","zeyodb")
                              .option("sfSchema","zeyoschema")
                              .option("sfRole","ACCOUNTADMIN")
                              .option("sfWarehouse","ZEYOWH")
                              .option("dbtable","zeyoin").load()
            /* either provide table in options or provide query like below */
                              .option("query","""
                                        select * from zeyodb.zeyoschema.zeyoin where BATCHID in
                                         (
                                         (select max(BATCHID) from zeyodb.zeyoschema.zeyoin),
                                         (select max(BATCHID)-1 from zeyodb.zeyoschema.zeyoin)
                                         )
                                         """).load()
```

Spark File Mode/ Write file in Spark

There are 4 modes → error(default), append, overwrite, ignore

```
=====================================================================
Write new data in existing folder but remove all old data files
=====================================================================
object obj {
 def main(args: Array[String]): Unit = {
   val conf = new SparkConf().setAppName("practice").setMaster("local[*]")
   val sc = new SparkContext(conf)
   val spark = SparkSession.builder().getOrCreate()

   val data = spark.read.orc("file:///home/Practice/data/devices.csv")
   data.createOrReplaceTempView("dataView")
   val filterData = spark.sql("SELECT * from dataView where lat>40")

   /* 1st way to create write orc file */
   filterData.write.mode("overwrite").orc("file:///home/Practice/data/d1")

   /* 2nd way to create write orc file */
   filterData.write.format("orc")
     .mode("overwrite")
     .save("file:///home/Practice/data/d2")

 }
}
```

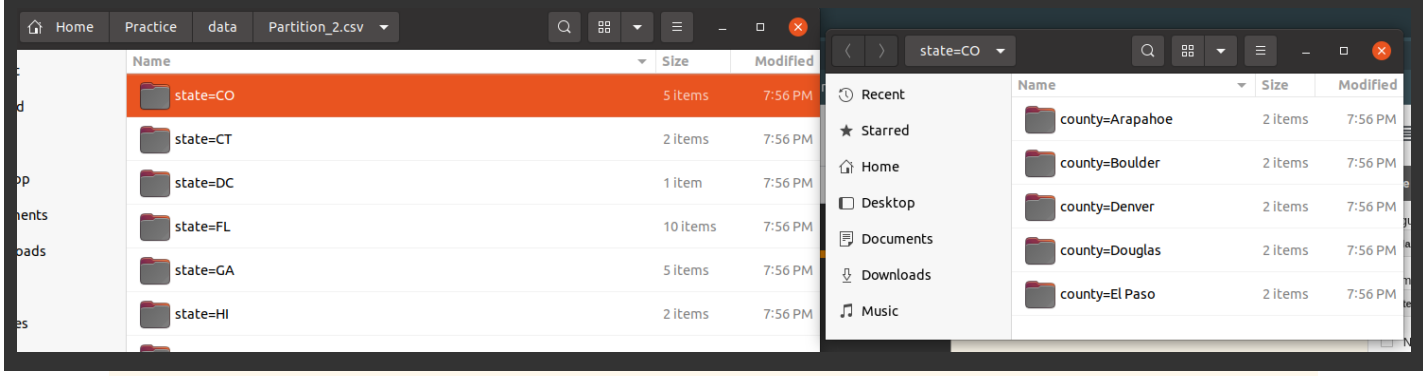
```
=====================================================================
Write/Add new data in existing folder but also keep all old data
=====================================================================
filterData.write.format("orc")
     .mode("append")
     .save("file:///home/Practice/data/d2")
```

```
=====================================================================
Write new data in folder but if folder already exists don't write data
=====================================================================
```

```
filterData.write.format("orc")
     .mode("ignore")
     .save("file:///home/Practice/data/d2")
```

Create Partitioned data using scala?

```
/* Read file form Local */
val df = spark.read.format("csv").option("header", true)
 .load("file:///home/Practice/data/usdata.csv")

/* data Partition by 'state' */
df.write.format("csv").partitionBy("state")
 .mode(SaveMode.Overwrite).save("file:///home/Practice/data/Partition_1.csv")

/* data Partition by 'state' and "country" */
df.write.format("csv").partitionBy("state", "county")
 .mode(SaveMode.Overwrite).save("file:///home/Practice/data/Partition_2.csv")
```



Options use cases

- **header - will enable headers to display in the table otherwise, it will be treated as one of the rows of data**
- **Delimiter - it will split data from that '~' into columns**

```
val csvdf = spark.read.format("csv")
            .option("header","true")
            .option("delimiter","~")
            .load("file:///C:/data/txns_head_pipe")
```

- **multiline - In order to process the JSON/CSV file with values in rows scattered across multiple lines, use option("multiLine", true)**

```
val df = spark.read
    .option("header",true)
    .option("multiLine",true)
    .csv("src/main/resources/address-multiline.csv")

df.show()
```

For example read below json -

```
{
        "id": 1,
        "TrainerName": "Zeyo",
        "Technology": "BigData",
        "address": {
                "temporary": "chennai",
                "permanent": "hyderabad"
        }
}
```

- **Read above json without 'multiline' option - will get error**

```
val df = spark.read.format("json").option("header", true)
                .load("file:///home/Practice/data/cm.json")
```

```
Exception in thread "main" org.apache.spark.sql.AnalysisException:
Since Spark 2.3, the queries from raw JSON/CSV files are disallowed when the
referenced columns only include the internal corrupt record column
(named _corrupt_record by default). For example:
```

- **Read above json with 'multiline' option -**

```
val df = spark.read.format("json").option("header", true)
            .option("multiline", true)
            .load("file:///home/Practice/data/cm.json")
```

Now code will successfully run

Spark SQL

1. Read data from the file and save it in variable
2. Create SQL view (the view is like a table for temporary to execute options that should not affect the original table)
3. Run SQL command using spark.sql(command) on that view just like a table.

```
val jsondf = spark.read.format("json")
.load("file:///C:/data/devices.json")

jsondf.show()


jsondf.createOrReplaceTempView("jdf")

val finaldf = spark.sql("select * from jdf where lat>40")

finaldf.show()
```

Spark DSL(Domain specific language)

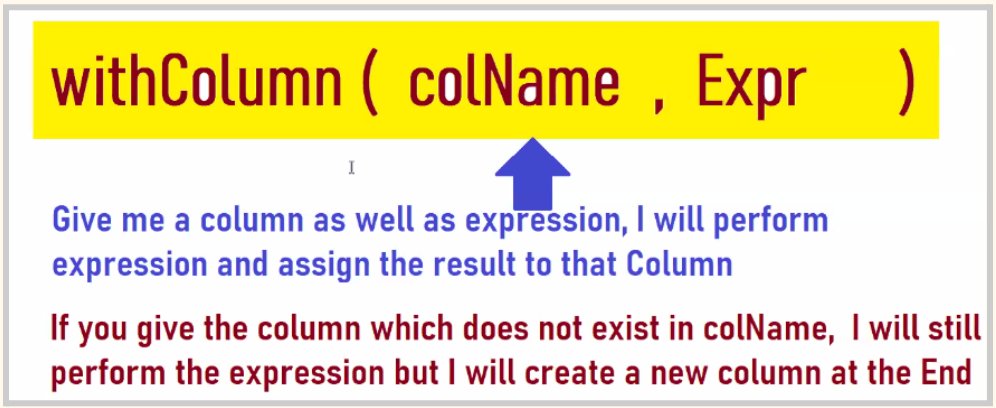
Filters with column conditions and operations

To run any operation over any column value, use col(columnName) to take columns from data.

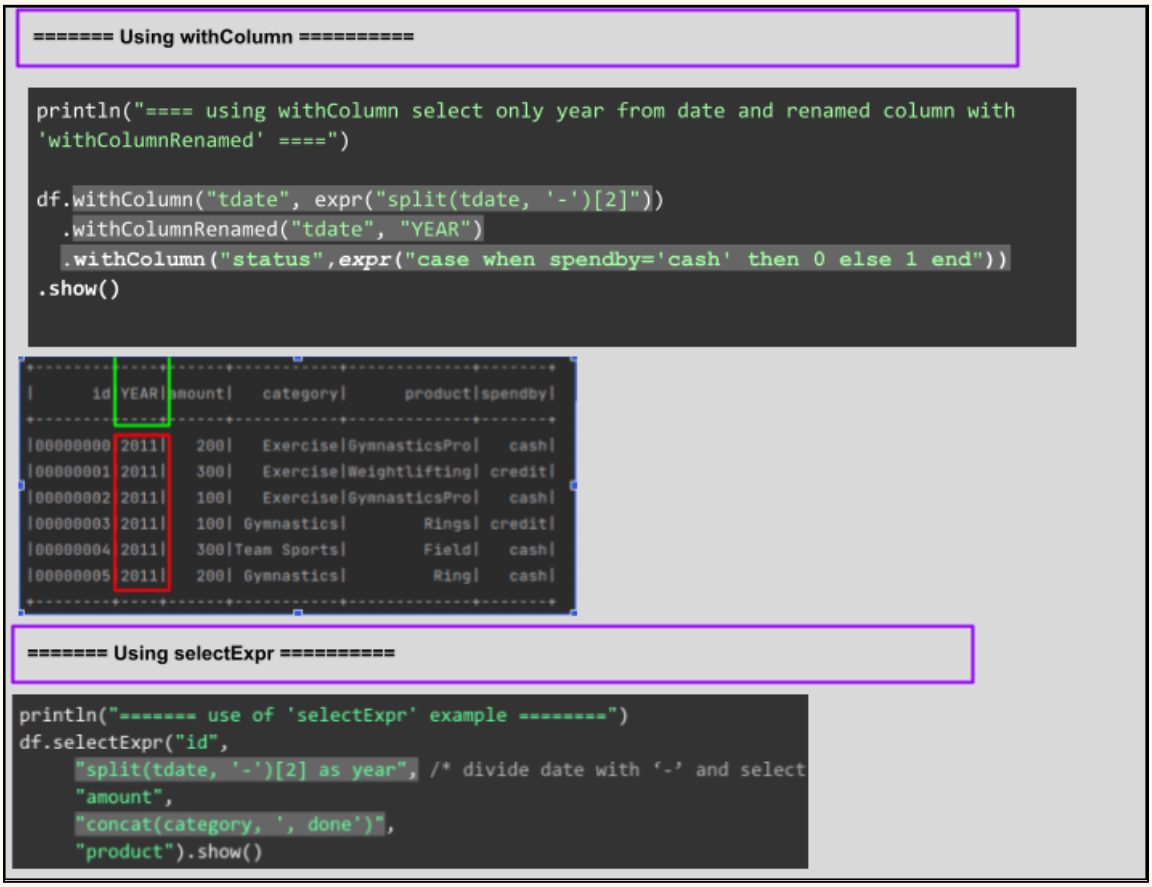
```
val df = spark.read.format("csv").option("header", value = true)
 .load("file:///D:/BigData/data/dt.txt")

println("======= Or '||' filter ========")
df.filter(col("category") === "Exercise"
        || col("spendby") === "cash").show()

println("======= And '&&' filter ========")
df.filter(col("category") === "Exercise"
        && col("spendby") === "cash").show()

println("======= Multi value filter ========")
df.filter(col("category") isin ("Exercise", "Gymnastics")).show()
println("======= not in Multi value filter ========")
df.filter(!(col("category") isin ("Exercise", "Gymnastics"))).show()

println("======= 'like' operator using filter ========")
df.filter(col("product") like "%Gymnastics%").show()
```

```
println("======= 'null' value filter ========")
df.filter(col("id") isNull).show()

println("======= not 'null' value filter ========")
df.filter(col("id") isNotNull).show()

//SQL Expression
df.filter("gender == 'M'").show(false)

//Array condition - array_contains() is inbuilt SQL pack function
df.filter(array_contains(df("languages"),"Java")).show(false)

//Struct condition
df.filter(df("name.lastname") === "Williams").show(false)

-------------------------------------
Few below alternatives for filter the column where “state” is column name
-------------------------------------

df.filter('state’ === "OH").show()
df.filter($state === "OH").show()
df.filter(col("state") === "OH").show()
df.where(df("state") === "OH").show()
df.where('state’ === "OH").show()
df.where($state === "OH").show()
df.where(col("state") === "OH").show()
```

withColumn use case



- To want to run the expression on only one column and without bothering other columns use this method, as the result expression will run on only that column and all other columns will also be in the result with any change.
- For example, using selectExpr() we have to give all column names that exist in the table with the expression column, using withColumn we can reduce that work.



- If in **colName** value **'column'** exist in the table then the expression value will override **otherwise if the column does not exist but the expression is correct new column will be created at the end of the table with the expressions value**, like below -

```
println("==== using withColumn when column not exist ====")

df.withColumn("nonExistColumn", expr("split(tdate, '-')[2]")).show()
```

```
==== using withColumn when column not exist ====
+--------+----------+------+-----------+-------------+--------+--------------+
|      id|     tdate|amount|   category|      product|spendby|nonExistColumn|
+--------+----------+------+-----------+-------------+--------+--------------+
|00000000|06-26-2011|   200|   Exercise|GymnasticsPro|   cash|          2011|
|00000001|05-26-2011|   300|   Exercise|Weightlifting| credit|          2011|
|00000002|06-01-2011|   100|   Exercise|GymnasticsPro|   cash|          2011|
|00000003|06-05-2011|   100| Gymnastics|        Rings| credit|          2011|
|00000004|12-17-2011|   300|Team Sports|        Field|   cash|          2011|
|00000005|02-14-2011|   200| Gymnastics|         Ring|   cash|          2011|
+--------+----------+------+-----------+-------------+--------+--------------+
```

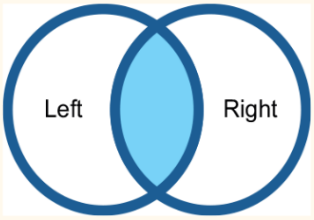
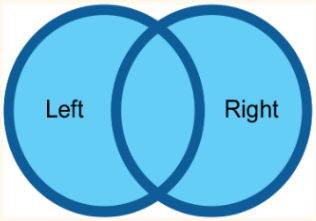
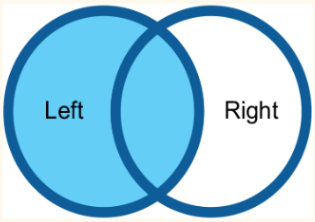
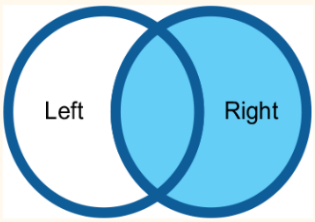
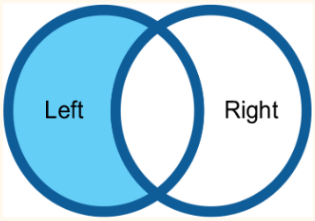
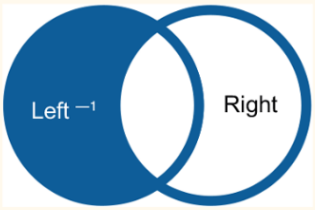
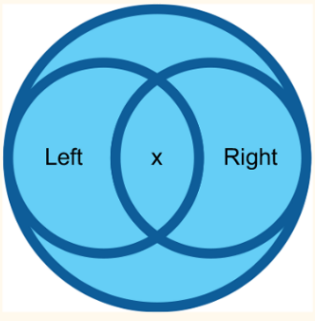
DSL Functions Queries

```
println("=== GroupBy and Aggregate method use ====")

/* find total amount of category */
df.groupBy("category")
 .agg(sum("amount").as("Total Amount"))
 .orderBy(col("category").desc)
 .show()

/* find count of category */
df.groupBy("category")
 .agg(count("category").as("count"))
 .orderBy(col("category").desc)
 .show()
```

Joins

Inner, Left, Right, Full joins, left_anti joins

Table 12.9 Join types in Spark

Join type	Aliases	Description	
Inner		Default type of join. Selects all rows from the left dataset and the right dataset, where the join condition is met.	
Outer	`full`, `fullouter`, `full_outer`	Selects data from both datasets based on the join condition and adds `null` when data is missing from the left or right.	
Left	`leftouter`, `left_outer`	Selects all rows from the left dataset, as well as all rows from the right dataset for which the join condition is met.	
Right	`rightouter`, `right_outer`	Selects all rows from the right dataset, as well as all rows from the left dataset for which the join condition is met.	
Left-semi	`left_semi`	Selects rows from only the left dataset for which the join condition is met.	
Left-anti	`left_anti`	Selects rows from only the left dataset for which the join condition is *not* met.	
Cross		Performs a Cartesian join of both datasets. As a reminder, a *Cartesian join* (also sometimes called a *Cartesian product*) is a join of every row of one table to every row of another table. As an example, if table `institution` with 100 rows is joined with table `subject` with 1,000 rows, the cross-join will return 100,000 rows.	

```scala
val df1 = spark.read.format("csv").option("header", "true")
 .load("file:///home/Practice/data/join1.csv")

val df2 = spark.read.format("csv").option("header", "true")
 .load("file:///home/Practice/data/join2.csv")

println("====== Inner Join ============")
/* --if both table have different column then compare like below  --------*/
df1.join(df2, df1("txnno") === df2("tno"), "inner")
 .drop("tno")
 .show()

/* ------if both table have same column just provide column name  --------*/
df1.join(df2, "txnno", "inner")
 .drop("tno")
 .show()

println("====== Left Join ============")
df1.join(df2, df1("txnno") === df2("tno"), "left")
 .drop("tno")
 .show()

println("====== Right Join ============")
df1.join(df2, df1("txnno") === df2("tno"), "right")
 .drop("txnno")
 .select("tno", "txndate", "amount", "custno", "spendby")
 .show()

println("====== Full Join ============")
df1.join(df2, df1("txnno") === df2("tno"), "full")
 .withColumn("txnno", expr("case when txnno is null then tno else txnno end"))
 .drop("tno")
 .show()

println("====== Left-Anti Join ============")
df1.join(df2, df1("txnno") === df2("tno"), "left_anti")
 .drop("tno")

println("====== Left-semi Join ============")
df1.join(df2, seq("txnno"), "leftsemi")

println("===== Join two tables when comparing more than one columns values===")
df1.join(df2, df1("txnno") === df2("tno") && df1("name") === df2("name"), "left_anti")
 .drop("tno")

println("===Union Join ===")
df1.union(df2)

println("=== Cross Join ===")
```

```
df1.crossJoin(df2)
```

Window Functions

```
import org.apache.spark.sql.functions._
```

Spark Window functions are used to calculate results such as the rank, row number e.t.c over a range of input rows. To perform an operation on a group, first we need to partition the data using `Window.partitionBy()`. Spark SQL supports three kinds of window functions:

1. ranking functions
2. analytic functions
3. aggregate functions

Ranking functions

```
===== Example of dense_rank() ====================
Task Find second highest salary department-wise, refer
below csv data
(using dense()_mark we can provide ran to each row)
```

dept	salary
dp3	450
dp1	250
dp2	500
dp2	700
dp1	200
dp3	650
dp1	900

```
=================================================
val partitionedData = Window.partitionBy(col("dept"))
 .orderBy(col("salary").desc)

df.withColumn("rank", dense_rank() over partitionedData)
 .filter("rank=2")
 .drop(col("rank"))
 .show()

/* if need to cast salary to INT use below code */
df.withColumn("salary", col("salary").cast("int"))
 .withColumn("rank", dense_rank() over partitionedData)
 .filter("rank=2")
 .drop(col("rank"))
 .show()
```

```
==============================
Output (All rows are ranked as per
department)
==============================
+----+------+----+
|dept|salary|rank|
+----+------+----+
| dp1|   900|   1|
| dp1|   250|   2|
| dp1|   200|   3|
| dp2|   700|   1|
| dp2|   500|   2|
| dp3|   650|   1|
| dp3|   450|   2|
+----+------+----+
```

```
==== Example of rank() ====================
```

rank() window function is used to provide a rank to the result within a window partition. This function leaves gaps in rank when there are ties. For example there are two salaries values are same, then rank() will assign same rank value to them, suppose **3 rank** and then for next row it will provide rank **-5(not 4)**

```
+-------------+----------+------+----+
|employee_name|department|salary|rank|
+-------------+----------+------+----+
|        James|     Sales|  3000|   1|
|        James|     Sales|  3000|   1|
|       Robert|     Sales|  4100|   3|
|         Saif|     Sales|  4100|   3|
|      Michael|     Sales|  4600|   5|
|        Maria|   Finance|  3000|   1|
|        Scott|   Finance|  3300|   2|
|          Jen|   Finance|  3900|   3|
|        Kumar| Marketing|  2000|   1|
|         Jeff| Marketing|  3000|   2|
+-------------+----------+------+----+
```

```
==== Example of row_number() ====================
```

`row_number():` Returns the rank of rows within a window partition.

```
==== Example of percent_rank() ====================
df.withColumn("rank", percent_rank() over partitionedData)
  .show()
```

```
+----+------+----+
|dept|salary|rank|
+----+------+----+
| dp1|   900| 0.0|
| dp1|   250| 0.5|
| dp1|   200| 1.0|
| dp2|   700| 0.0|
| dp2|   500| 1.0|
| dp3|   650| 0.0|
| dp3|   450| 1.0|
+----+------+----+
```

Processing Complex data

There are 3 types of complexities -

Complexity type	How to Process	How to Generate

1. **Multiline**	df.option(“**multiline”, true**)	
2. **Struct data**	**. (dot)**	**struct**
3. **Array**	**explode()**	**collect_list(col(“”)).as(“”)**

```
--- Refer Below json example of struct data -----------------
Practice #1 -
```

```
{
        "id": 1,
        "TrainerName": "Zeyo",
        "Technology": "BigData",
        "address": {
                "temporary": "chennai",
                "permanent": "hyderabad"
        }
}
```

```
--------------------------------------------------

/* === complex data processing ===*/
val df = spark.read.format("json")
 .option("header", true)
 .option("multiline", true)
 .load("file:///home/Practice/data/cm.json")

/* ======== normal result  ============= */
df.show()
df.printSchema()

/* ===== flatten complex hierarchical data using .(dot) ====== */
val finalDf = df.select(
               "Technology",
               "TrainerName",
              "address.permanent",
             "address.temporary",
            )

finalDf.show()
finalDf.printSchema()

/*= another way to flatten hierarchical data using .(dot) & (*) ==*/
val finalDf = df.select(
               "Technology",
               "TrainerName",
              "address.*"
```

```
--------------------------------------
df.show()
df.printSchema()    result
--------------------------------------
+----------+-----------+--------------------+---+
|Technology|TrainerName|             address| id|
+----------+-----------+--------------------+---+
|   BigData|       Zeyo|{hyderabad, chenn...|  1|
+----------+-----------+--------------------+---+

root
 |-- Technology: string (nullable = true)
 |-- TrainerName: string (nullable = true)
 |-- address: struct (nullable = true)
 |    |-- permanent: string (nullable = true)
 |    |-- temporary: string (nullable = true)
 |-- id: long (nullable = true)


--------------------------------------
finalDf.show()
finalDf.printSchema()    result
--------------------------------------
+----------+-----------+---------+---------+
|Technology|TrainerName|permanent|temporary|
+----------+-----------+---------+---------+
|   BigData|       Zeyo|hyderabad|  chennai|
+----------+-----------+---------+---------+


root
 |-- Technology: string (nullable = true)
 |-- TrainerName: string (nullable = true)
 |-- permanent: string (nullable = true)
 |-- temporary: string (nullable = true)
```

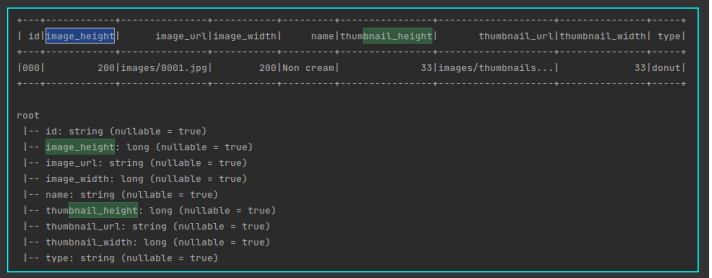
```
          )
```

```
----Struct example ----------------------------------------
Practice #2 -  processing complex data when data have ambiguity
```

```
{
        "id": "000",
        "type": "donut",
        "name": "Non cream",
        "image": {
                "url": "images/0001.jpg",
                "width": 200,
                "height": 200
        },
        "thumbnail": {
                "url": "images/thumbnails/0001.jpg",
                "width": 33,
                "height": 33
        }
}
```

```
-------------------------------------------------
    /* normally process above data */
      val normalJson = df.select(
           "id",
           "image.height",
           "image.url",
           "image.width",
           "name",
           "thumbnail.height",
           "thumbnail.url",
           "thumbnail.width",
           "type"
         )
      normalJson.show()
      normalJson.printSchema()

/*resolve ambiguity issue with providing alias name to them */
   val flatttenJson = df.selectExpr(
     "id",
     "image.height as image_height",
     "image.url as image_url",
     "image.width as image_width",
     "name",
     "thumbnail.height as thumbnail_height",
     "thumbnail.url as thumbnail_url",
     "thumbnail.width as thumbnail_width",
     "type"
   )
```

```
---------------------------------------
normalJson.show()
normalJson.printSchema()    result
---------------------------------------

See below table result, both image.url and thumbnail.url have column name 'url'
whoch is ambiguity problem
+---+------+---------------+-----+---------+------+--------------------+-----+-----+
| id|height|            url|width|     name|height|                 url|width| type|
+---+------+---------------+-----+---------+------+--------------------+-----+-----+
|000|   200|images/0001.jpg|  200|Non cream|    33|images/thumbnails...|   33|donut|
+---+------+---------------+-----+---------+------+--------------------+-----+-----+

root
 |-- id: string (nullable = true)
 |-- height: long (nullable = true)
 |-- url: string (nullable = true)
 |-- width: long (nullable = true)
 |-- name: string (nullable = true)
 |-- height: long (nullable = true)
 |-- url: string (nullable = true)
 |-- width: long (nullable = true)
 |-- type: string (nullable = true)

-
-------------------------------------------------
flattenJson.show()
flattenJson.printSchema()    result
-------------------------------------------------------------
```



```scala
/* second way to process ambiguity data using select() or 'withColumn()' */
   val flatjsonSecondWay = df.select(
     col("id"),
     col("name"),
     col("type "),
     col("image.height").as("image_height"),
     col("image.url").as("image_url"),
     col("image.width").as("image_width"),
     col("thumbnail.height").as("thumbnail_height"),
     col("thumbnail.url").as("thumbnail_url"),
     col("thumbnail.width").as("thumbnail_width")
   )

      flatttenJson.show()
      flatttenJson.printSchema()
```

---- example of Array data ----------

```
{
        "Name": "Test",
        "Mobile": 12345678,
        "status": true,
        "Pets": ["Dog", "cat"],
        "Address": {
                "Permanent address": "USA",
                "current Address": "AU"
        }
}
```

```scala
------------------------------------------------------------
df.printSchema()

/* Process Array type complex data using explode() with select() */
val flatJson = df.select(
 col("Address.*"),
 col("Mobile"),
 col("Name"),
 explode(col("Pets")).as("Pets"),
 col("status")
)

flatJson.show()
flatJson.printSchema()

/* 2nd way to - Process Array complex data using explode() and 'withColumn()' */
val flatJson = df.withColumn("Pets", explode(col("Pets")))
 .withColumn("Permanent Address", col("Address.`Permanent Address`"))
```

---

df.printSchema() result

```
root
 |-- Address: struct (nullable = true)
 |    |-- Permanent address: string (nullable = true)
 |    |-- current Address: string (nullable = true)
 |-- Mobile: long (nullable = true)
 |-- Name: string (nullable = true)
 |-- Pets: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- status: boolean (nullable = true)
```

---

flatJson.printSchema() result

```
 .withColumn("Current Address", col("Address.`current Address`"))
 .drop("Address")

flatJson2.show()
flatJson2.printSchema()

/* Generate Array complex data using collect_list() */
val cjson = flatJson.select(
   struct(
     col("`Permanent Address`"),
     col("`current Address`")
   ).as("Address"), /* make address back to struct */
  col("Mobile"),
  col("Name"),
  col("status"),
  col("pets")
).groupBy("Mobile", "status", "Name", "Address") /* groupBy common data */
 .agg(collect_list("pets").as("pets"))  /* do collect_list() for array */
```

```
+-----------------+---------------+--------+----+----+------+
|Permanent address|current Address|  Mobile|Name|Pets|status|
+-----------------+---------------+--------+----+----+------+
|              USA|             AU|12345678|Test| Dog|  true|
|              USA|             AU|12345678|Test| cat|  true|
+-----------------+---------------+--------+----+----+------+

root
 |-- Permanent address: string (nullable = true)
 |-- current Address: string (nullable = true)
 |-- Mobile: long (nullable = true)
 |-- Name: string (nullable = true)
 |-- Pets: string (nullable = true)
 |-- status: boolean (nullable = true)

-------------------------------------------------
Generate complex data from flatten data --> result
-------------------------------------------------
root
 |-- Mobile: long (nullable = true)
 |-- status: boolean (nullable = true)
 |-- Name: string (nullable = true)
 |-- Address: struct (nullable = false)
 |    |-- Permanent Address: string (nullable = true)
 |    |-- current Address: string (nullable = true)
 |-- pets: array (nullable = false)
 |    |-- element: string (containsNull = false)
```

```
----  example of Array contains struct data --------------------
Data structure is like below
```

```
-----------------------------------
process(flatten) data result
-----------------------------------
```

```
|-- data: array (nullable = true)
|    |-- element: struct (containsNull = true)
|    |    |-- avatar: string (nullable = true)
|    |    |-- email: string (nullable = true)
|    |    |-- first_name: string (nullable = true)
|    |    |-- id: long (nullable = true)
|    |    |-- last_name: string (nullable = true)
|-- page: long (nullable = true)
|-- per_page: long (nullable = true)
|-- support: struct (nullable = true)
|    |-- text: string (nullable = true)
|    |-- url: string (nullable = true)
|-- total: long (nullable = true)
|-- total_pages: long (nullable = true)
```

```
/* -- Process(flatten) above(reqres.json) data -----  */
val flatJson = df.withColumn("data", explode(col("data")))
 .select(
   col("data.*"),
   col("page"),
   col("per_page"),
   col("support.*"),
   col("total"),
   col("total_pages")
 )

/* --- generate array complex data using 'select()'------ */
val cjson = flatJson.select(
 struct(
   col("avatar"),
   col("email"),
   col("id"),
   col("first_name"),
   col("last_name"),
 ).as("element"), /* create 'element' struct type */
 struct(
   col("text"),
   col("url")
 ).as("support"), /* create 'support' struct type */
 col("page"),
 col("per_page"),
 col("total"),
 col("total_pages"),
```

```
root
 |-- avatar: string (nullable = true)
 |-- email: string (nullable = true)
 |-- first_name: string (nullable = true)
 |-- id: long (nullable = true)
 |-- last_name: string (nullable = true)
 |-- page: long (nullable = true)
 |-- per_page: long (nullable = true)
 |-- text: string (nullable = true)
 |-- url: string (nullable = true)
 |-- total: long (nullable = true)
 |-- total_pages: long (nullable = true)
```

```
root
 |-- page: long (nullable = true)
 |-- per_page: long (nullable = true)
 |-- total: long (nullable = true)
 |-- total_pages: long (nullable = true)
 |-- support: struct (nullable = false)
 |    |-- text: string (nullable = true)
 |    |-- url: string (nullable = true)
 |-- data: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- avatar: string (nullable = true)
 |    |    |-- email: string (nullable = true)
 |    |    |-- first_name: string (nullable = true)
 |    |    |-- id: long (nullable = true)
 |    |    |-- last_name: string (nullable = true)
```

```
"per_page", "total", "total_pages", "support") /* group common
 value */
("element").as("data")) /* write all 'element' values in array

l", "avatar", "email", "id", "first_name", "last_name") /* drop



ate array complex data using 'withColumn()' */
atJson.withColumn("support",



t")
"per_page", "total", "total_pages", "support")



"),
"),
_name"),

ame")
```

Generating Complex data

Below is CSV data, now generate complex data using struct→

Technology	TrainerName	permanent	temporary	id	workloc
BigData	Zeyo	Hyderabad	Chennai	1	pune

```
val flatjson = df.select(
 col("Technology"),
 col("id"),
 struct(
   struct(
     col("permanent"),
```

```
    col("temporary"),
    col("workloc")
   ).as("user"),
   col("TrainerName")
 ).as("address")
)

flatjson.show()
flatjson.printSchema()
-----------------------------------------------------------------------

Save above result in file and complex json will be generated -
{"Technology":"BigData","id":"1","address":{"user":{"permanent":"Hyderabad","temporary":"Chennai","workloc":"pun
e"},"TrainerName":"Zeyo"}}
-----------------------------------------------------------------------
root
 |-- Technology: string (nullable = true)
 |-- id: string (nullable = true)
 |-- address: struct (nullable = false)
 |    |-- user: struct (nullable = false)
 |    |    |-- permanent: string (nullable = true)
 |    |    |-- temporary: string (nullable = true)
 |    |    |-- workloc: string (nullable = true)
 |    |-- TrainerName: string (nullable = true)

+----------+-----------+---------+---------+---+-------+
|Technology|TrainerName|permanent|temporary| id|workloc|
+----------+-----------+---------+---------+---+-------+
|   BigData|       Zeyo|Hyderabad|  Chennai|  1|   pune|
+----------+-----------+---------+---------+---+-------+
```

Tasks -

```
=========================================================================
Task #1. Read data.csv file and only print specific columns in dataframe
=========================================================================
/* read file */
val df = spark.read.format("csv").option("header", true)
 .load("file:///home/Practice/data/usdata.csv")

/* print only specific columns */
```

```
df.select("first_name", "last_name").show()
```

```
=====================================================================
Task #2. Select only year from date[01-01-2000] column value
  Sol =>  Using selectExpr, we can run any sql function over any column
=====================================================================
println("======= use of 'selectExpr' example ========")
df.selectExpr("id",
     "split(tdate, '-')[2] as year", /* divide date with '-' and select 2nd index */
     "amount",
     "concat(category, ', done')",
     "product").show()
```

```
=====================================================================
Task #3. Add new column and provide '0' or '1' as per spendBy column values
Using selectExpr,we can run any sql function over any column
=====================================================================
println("======= use of 'selectExpr' example ========")
df.selectExpr("id",
     "split(tdate, '-')[2] as year", /* divide date with '-' and select 2nd index */
     "amount",
     "concat(category, ', done')",
     "Product",
     "case when spendby = 'case' then 0 else 1 end as STATUS").show()
```

```
=====================================================================
Task #4 - Get Day of the week from Date and Timestamp columns
=====================================================================
df.withColumn("date", to_date(col("date"), "d/M/yyyy")) /* Converts the column into a `DateType` with a
specified format` */
 .withColumn("Day", date_format(col("date"), "EE")) /* Converts a date/timestamp/ to a value of string in the
format specified by the date format given by the second argument. */
 .groupBy("dispatching_base_number", "Day")
 .agg(sum("trips").as("Total Trips"))
 .orderBy(col("Total Trips"))
 .show()
```

```
=====================================================================
Task #5 - Find second highest salary department-wise, refer below csv data
(using dense()_mark we can provide ran to each row)
```

dept	salary
dp3	450
dp1	250
dp2	500
dp2	700
dp1	200
dp3	650
dp1	900

```
=====================================================================
val partitionedData = Window.partitionBy(col("dept"))
 .orderBy(col("salary").desc)

df.withColumn("salary", col("salary").cast("int"))  /* cast string column to int */
 .withColumn("rank", dense_rank() over partitionedData)
 .filter("rank=2")
 .drop(col("rank"))
 .show()
```

```
=====================================================================
Task #6 - Requirement:
   1. Select PROV_PH_DTL column from the entire dataset
   2. Remove everything after $ (including $)
   3. Split the column from '|'
   4. Now, you should have total 4 columns, now rename the columns as:
         a. PH_TY_CD
         b. PH_NUM
         c. PH_EXT_NUM
         d. PH_PRIM_IND
=====================================================================
df.show(5)
/* ==== show specific column ========*/
df.select("PROV_PH_DTL").show(5, false)

/* substring column value using select */
println("=== split data before '$' and then split with '|'  ========")
val prov_df = df.select(
    substring_index(col("PROV_PH_DTL"), "$", 1).as("PROV_PH_DTL")) /* get substring up to '$'*/
   .withColumn("PROV_PH_DTL", split(col("PROV_PH_DTL"), "\\|")) /* split value from '|'*/

prov_df.show(5, false)

/* Now column is divided into 4 values, convert them in new column and assign a column name */
prov_df.select(
 col("PROV_PH_DTL").getItem(0).as("PH_TY_CD"),
 col("PROV_PH_DTL").getItem(1).as("PH_NUM"),
 col("PROV_PH_DTL").getItem(2).as("PH_EXT_NUM"),
```

```
 col("PROV_PH_DTL").getItem(3).as("PH_PRIM_IND")
).show()
```

```
=====================================================================
Task #7 - SQL Scenario, refer below table 1 & table 2
```

Table 1 ⇒

```
+-----+-------+------+
|empno|empname|empman|
+-----+-------+------+
| 1001|   zeyo|10,101|
| 1002|   ravi|11,102|
+-----+-------+------+
```

Table 2 ⇒

```
+---+-------+
|mno|  mname|
+---+-------+
| 10|   Rani|
| 11| Vidyut|
|101| Mithra|
|102|akshara|
+---+-------+
```

Result should be

```
+-----+-------+------------------+
|empno|empname|collect_list(mname)|
+-----+-------+------------------+
| 1002|   ravi|  [Vidyut, akshara]|
| 1001|   zeyo|     [Rani, Mithra]|
+-----+-------+------------------+
```

```
=============================================================================
/* Explode employee data */
println("==== explode emp table ====")
val emp = df.withColumn("empman",
 explode(
   split(col("empman"), ",") /* split data from comma then cast to array */
     .cast("array<long>")
 ) /* then explode array data */
)

/* Inner join of 2 tables*/
println("==== Inner join emp and manager table ====")
val joinDF = emp.join(df2, emp("empman") === df2("mno"), "inner")
joinDF.show()

/* Generate complex data */
println("==== Generate complex data ====")
val finalDF = joinDF
 .select(
   "empno",
   "empname",
   "mname",
 )
 .groupBy("empno", "empname")
 .agg(collect_list("mname"))

finalDF.show()
finalDF.printSchema()
```

```
=====================================================================
Task #8 - Integrate Spark with Snowflake
   1. Sai You have a data sitting in Snowflake in table ‘zeyoin’
   2. integrate(connect) spark with Snowflake
   3. Read the data from snowflake table
   4. write to your Local with partition (itr-1 -- newdata)
   5. Use below detail to connect with snowflake ⇒

         • url – https://YI78860.ap-southeast-1.snowflakecomputing.com
         • account – YI78860
         • username – zeyoanalytics2
         • Password – Aditya908
         • Warehouse – ZEYOWH
         • Database – zeyodb
         • Schema – zeyoschema
         • Table – zeyoin
         • Role – ACCOUNTADMIN

   6. Next Day ⇒ In snowflake we got 3 Extra/new Rows with BatchID 2.
   7. Now Read yesterday's data , Today's data and create 3 folders.
         • Delete → (data with was in yesterday’s data but not in today’s data)
         • new (Updated) → row which is either new or data is updated(like Run Revision programsjson is
           updated)
         • existing → data which exists in both Today’s and yesterday’s
=====================================================================
package pack

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, udf}
import org.apache.spark.{SparkConf, SparkContext}
import spray.json._

object snowflakeIntegration {

 def compareJson_udf = udf((tj: String, yj: String) => {
   /* first parse string to json using (Spray maven dependency)*/
   val json1 = tj.parseJson
   val json2 = yj.parseJson
   val results = (json1 == json2)
   results
 })

 def main(args: Array[String]): Unit = {
   val conf = new SparkConf().setAppName("revision").setMaster("local[*]")
   val sc = new SparkContext(conf)
```

```
sc.setLogLevel("ERROR")
val spark = SparkSession.builder().getOrCreate()

/* Read data from snowflake database */
val snowdf = spark.read.format("snowflake")
  .option("sfURL", "https://YI78860.ap-southeast-1.snowflakecomputing.com")
  .option("sfAccount", "YI78860")
  .option("sfUser", "zeyoanalytics2")
  .option("sfPassword", "Aditya908")
  .option("sfDatabase", "zeyodb")
  .option("sfSchema", "zeyoschema")
  .option("sfRole", "ACCOUNTADMIN")
  .option("sfWarehouse", "ZEYOWH")
  .option("query",
    """select * from zeyodb.zeyoschema.zeyoin where BATCHID in
            (
            (select max(BATCHID) from zeyodb.zeyoschema.zeyoin),
            (select max(BATCHID)-1 from zeyodb.zeyoschema.zeyoin)

            )""").load()

snowdf.show(false)

/* As "BATCHID" increasing next day, so get today's and yesterdays 'BATCHID' */
val yesterdayBatchDF = snowdf.selectExpr("min(BATCHID) as BATCHID")
yesterdayBatchDF.show()

val todayBatchDF = snowdf.selectExpr("max(BATCHID) as BATCHID")
todayBatchDF.show()

/* save today's and yesterday's data into different dataframes */
val yesterdayData = snowdf.join(yesterdayBatchDF, Seq("BATCHID"), "inner").drop("BATCHID")
  .withColumnRenamed("VALUE", "YVALUE")
yesterdayData.show(false)

val todayData = snowdf.join(todayBatchDF, Seq("BATCHID"), "inner").drop("BATCHID")
  .withColumnRenamed("VALUE", "TVALUE")
todayData.show(false)

/* then join both today and yesterday df */
val joindf = yesterdayData.join(todayData, Seq("ID"), "full")
joindf.show(false)

/* now filter delete data - rows which now present in Today data, consider as delete */
val deleteData = joindf.filter(col("TVALUE").isNull)
```

```
      .select("id", "YVALUE")

    println("=== Existing ==")
    deleteData.show()

    /* now filter new data - rows which are newly added Today */
    val newData = joindf.filter(col("YVALUE").isNull)
      .select("id", "TVALUE")

    println("==== new ==")
    newData.show()

    /* now filter new data - rows which are newly added Today */
    val existing_Or_UpdatedData = joindf.filter(col("YVALUE").isNotNull
      && col("TVALUE").isNotNull)

    existing_Or_UpdatedData.show(false)

    /* compare yesterday "value" json and today "value" json and filter which data is updated and which one is
not */
    val jsonCompare = existing_Or_UpdatedData.withColumn("isEqual",
      compareJson_udf(existing_Or_UpdatedData("YVALUE"), existing_Or_UpdatedData("TVALUE")))

    println("==== if json compare is true the, means data is Existing ==")
    val existing = jsonCompare.filter(col("isEqual") === true)
      .select("ID", "TVALUE")

    existing.show(false)

    println("==== if json compare is true the, means data is Updated ==")
    val updated = jsonCompare.filter(col("isEqual") === false)
      .select("ID", "TVALUE")

    updated.show(false)

    /* union both new and updated data into one dataframe */
    val finalNew = newData.union(updated)
    finalNew.show()
  }
}
```

```
================================================================
Task #9 - Run Revision programs
================================================================
package test
```

```scala
import org.apache.spark.sql.functions.{col, column, expr, sum}
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}

object obj1 {

 case class schema(id: Int, date: String, cus_no: String, amount: String, category: String, product: String,
city: String, state: String, spendby: String)

 def main(args: Array[String]): Unit = {

   val conf = new SparkConf().setAppName("Revision").setMaster("local[*]")
   val rdd = new SparkContext(conf)
   val spark = SparkSession.builder().getOrCreate()
   import spark.implicits._
   rdd.setLogLevel("error")

   /* ===== Create a scala List with 1,4,6,7 and Do an iteration and add 2 to it =======*/
   val int_list = List(1, 2, 3, 4, 5, 6)
   println("\n---- Do an iteration over integer list and add 2 to each item ---")
   int_list.map(x => x + 2)
     .foreach(println)

   /* ===== Create a scala List with zeyobron,zeyo and analytics and filter elements contains zeyo =======*/
   val string_list = List("zeyobron", "bron", "zeyo", "saiZeyoAdity", "sai")
   println("\n---- filter elements contains zeyo ----")
   string_list.filter(x => x.contains("zeyo"))
     .foreach(println)


   println("\n===== Read file1 as an rdd and filter gymnastics rows =======")
   val file1_data = rdd.textFile("file:///home/Practice/data/file1.txt")
   file1_data.filter(x => x.contains("Gymnastics"))
     .take(5).foreach(println)

   println("\n--- Create a case class and Impose case class to it for schema rdd " +
     "And filter product contains Gymnastics and " +
     "then convert it to Dataframe -----")
   val gymnastics_df = file1_data.map(x => x.split(","))
     .map(x => schema(x(0).toInt, x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8)))
     .filter(x => x.product.contains("Gymnastics"))

   /* convert rdd to df using schema RDD */
```

```
gymnastics_df.toDF().show(5)

println("\n--- convert data to Row Rdd then convert rdd to Dataframe -----")
val gymnastics_df2 = file1_data.map(x => x.split(","))
  .map(x => Row(x(0).toInt, x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8)))

val structSchema = StructType(Array(
  StructField("id", IntegerType, true),
  StructField("date", StringType, true),
  StructField("cus_no", StringType, true),
  StructField("amount", StringType, true),
  StructField("category", StringType, true),
  StructField("product", StringType, true),
  StructField("city", StringType, true),
  StructField("state", StringType, true),
  StructField("spendby", StringType, true),
))

spark.createDataFrame(gymnastics_df2, structSchema).show(5)

println("\n--- Read CSV file, set header 'true' and print -----")
spark.read.format("csv").option("header", true)
  .load("file:///home/Practice/data/file3.txt")
  .show(5)

println("\n--- Read json file and print -----")
var json_df = spark.read.format("json").load("file:///home/Practice/data/file4.json")
json_df.show(5)

println("\n--- Read Parquet file and print -----")
var parquet_xml = spark.read.format("parquet").load("file:///home/Practice/data/file5.parquet")
parquet_xml.show(5)

println("\n--- Read XML file and print -----")
var xml_df = spark.read.format("xml")
  .option("rowTag", "txndata")
  .load("file:///home/Practice/data/file6")
xml_df.show(5)

println("\n------ Define a unified column list and impose using select for all the dataframe " +
  "and Union all the dataframes ----")

/* first create list of all column in a table*/
val column_list = List(
  "txnno",
```

```
  "txndate",
  "custno",
  "amount",
  "category",
  "product",
  "city",
  "state",
  "spendby"
)

/* second - Spark Select with a List of Columns Scala */
json_df = json_df.select(column_list.map(column): _*)

/* third step - Union other df */
val union_df = json_df.union(xml_df)
  .union(parquet_xml)
union_df.show(5)

val data = union_df.withColumn("txndate", expr("split(txndate, '-')[2]"))
  .withColumnRenamed("txndate", "YEAR")
  .withColumn("STATUS", expr("CASE WHEN spendby='cash' THEN 0 ELSE 1 END"))
  .filter(col("txnno") > 50000)
data.show(5)

println("\n---- Find the Cumulative sum of amount foreach category -----")
data.groupBy("category")
  .agg(sum("amount").as("Total amount"))
  .show(10)

println("\n--- Write as an avro in local with mode Append and partition the category column ---")
data.write.format("avro")
  .mode(SaveMode.Append)
  .partitionBy("category", "spendby")
  .save("file:///home/Practice/data/avroRevisionData")


println("\n--- Join join1.csv and join2.csv find the common ids in both data and" +
  " find a way to remove the ids of second dataframe in the first data frame ---")

val df1 = spark.read.format("csv").option("header", true).load("file:///home/Practice/data/join1.csv")
val df2 = spark.read.format("csv").option("header", true).load("file:///home/Practice/data/join2.csv")

df1.join(df2, df1("txnno") === df2("tno"), "inner")
  .drop("tno")
  .show()
```

```
 }
}
```

Scala Technical Questions

1. Difference between sc.textfile() and sc.wholeTextFiles()?
   sc.textfile() →Each row will be a record
   sc.wholeTextFiles() → read the entire content of a file as a single record

```
val readDataUsingTextFile = sc.textFile("file:///home/Practice/data/data2.txt")
val readDataUsingWholeTextFiles = sc.wholeTextFiles("file:///home/Practice/data/data2.txt")
val result = readDataUsingTextFile.map(x => (x,1))
val result2 = readDataUsingWholeTextFiles.map(x => (x,1))

println("==== result using text file ======")
result.take(2)foreach(println)  // '1' is added after every row record
println("==== result using whole text file ======")
result2.take(2).foreach(println) // "1" is added after all content of file

------------------------------------------------------------------------
sc.textFile() result      ||         sc.wholeTextFiles() result
--------------------------||-----------------------------------------
                          ||  ((file:/home/qualitest/Practice/data/data2.txt,ApacheJMeter
                          ||  KatalonStudio
(ApacheJMeter,1)          ||  AI
(KatalonStudio,1)         ||  ),1)
```

2. Function return type is Unit, what does it mean?

   Unit type is the same as void in JAVA, which means the function doesn't return anything

3. What is closure?

   Closure is like a Function whose return type is defined on the expression. Closure can be defined in any scope (class, Object, function)

```
var a = (i:Int, j:Int) => i * j       /* return type is Int */
var a2 = (i:Int, j:Int) => "Hello"    /* return type is String */
var a3 = (i:Int, j:Int) => List("Hello", "world", "Ayushi")   /* return type is List */
```

4. What is String Interpolation?

   i. **The 's' String Interpolator**

   The literal 's' allows the usage of variable directly in processing a string, when you prepend 's' to it

```
val name = "James"
println(s "Hello, $name") //output: Hello, James
```

   ii. **The ' f ' Interpolator**

   The literal 'f' interpolator allows to create a formatted String, similar to printf in C language

```
val height = 1.9d
val name = "James"
println(f"$name%s is $height%2.2f meters tall") //James is 1.90 meters tall
```

   iii. **'raw' Interpolator**

   The 'raw' interpolator is similar to 's' interpolator except that it performs no escaping of literals within a string.

```
object Demo {
   def main(args: Array[String]) {
      println(raw"Result = \n a \n b")
   }
}
```

Output –

```
Result = \n a \n b
```

5. Exceptional Handling in Scala?

```
object Demo {
   def main(args: Array[String]) {
      try {
         val f = new FileReader("input.txt")
      } catch {
         case ex: FileNotFoundException => {
            println("Missing file exception")
         }

         case ex: IOException => {
            println("IO Exception")
         }
      } finally {
         println("Exiting finally...")
      }
   }
}
```

6. What is a Match expression/Pattern Matching?

   It is the same as the Switch case in Java.

```
def matchTest(x: Int): String = x match {
  case 1 => "one"
  case 2 => "two"
  case _ => "many"   /* mend of pattern match*/

  print(  "this is done")
  return String
}
```

```scala
def main(args: Array[String]) {
   val age = 100;

   age match {
     case 20 => println(age);
     case 18 => println(age);
     case 30 => println(age);
     case 40 => println(age);
     case 50 => println(age);

     case _ => println("default");
   }
```

```scala
  val i = 7;
  i match {
    case 1 | 3 | 5 | 7 | 9 => println("odd");
    case 2 | 4 | 6 | 8| 10 => println("even");
```

7. Basic ways to write functions in Scala?

```scala
def add(x: Int, y: Int): Int = {
  return x + y;
}

def subtract(x: Int, y: Int): Int = {
  x - y;
}

def multiply(x: Int, y: Int): Int = x * y;

def divide(x: Int, y: Int) =  x / y;

def main(args: Array[String]) {
  println(add(45, 15));
}
```

8. What are Anonymous functions in Scala?

Without giving any name to a function we can assign it to a **Variable**, it's called Anonymous function.

```
/*   ======== Anonymous Functions  ========== */
val anonymous_functions = (x: Int, y: Int) => x + y
/* call variable and pass arguments */
println("Result of Anonymous function - " + anonymous_functions(500, 100))
```

```
Practice › main(args: Array[String])
scalaPractice ×
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ...
Result of Anonymous function - 600
```

9. What are Higher order functions?

Higher order functions are those functions which are able to **take functions as arguments** and are able to **return functions as result**.

```
object scalaPractice {
  /*   ======== higher-order Functions  ========== */
  def higherOrder_functions(x: Int, y: Int, f: (Double, Double) => Double): Double = {
    f(x, y)
  }

  def main(args: Array[String]): Unit = {
    /*   ======== call higher-order Functions, result: 5000.0  ========== */
    println("Result of Anonymous function - " + higherOrder_functions(50, 100, (x,y) => x * y))
  }
}
```

10. Define Array in scala?

```
/* 1st way using new Keyword */
val arr1: Array[Int] = new Array[Int](5)
arr1(0) = 10
arr1(1) = 10
arr1(2) = 10

/* 2nd way to create array by direct initialize */
val arr2: Array[Int] = Array(1, 2, 3, 4, 5)
```

11. How to concat two Arrays in Scala?

```
  /* Concat two Arrays */
  import Array.concat
  val result = concat(arr1, arr2)
}
```

12. How to define **For Loop** in Scala?

```
/* 1st way loop within range */
/* for(initialize <- range) */
for(i <- 1 ≤ to ≤ 5) {
  println("number:" + i)
}

/* another way to define loop using dot(.) */
for(i <- 1 ≤ .to( ≤ 5)) {
  println("number:" + i)
}

/* 2nd way loop until range, */
for(i <- 1 ≤ until  < 5) {
  println("number:" + i)
}

/* 3rd way loop with multiple ranges, below variable j range will work as nested loop */
for(i <- 1 ≤ until  < 5 ; j <- 1 ≤ to ≤ i) {
  println("number i:" + i + ", number j:" + j)
}

/* 4th way with List */
var list = List(1,2,3,4,5)
for(i <- list) {
  println("number:" + i)
}

/* 5th way with Array */
val arr = Array(10,20,30,40,50)
for (i <- 0 ≤ to ≤ (arr.length-1)) {
  println(arr(i))
}
```

```
/* 6th way only go inside loop if condition satisfy */
for (i <- range(1,5); if i > 2) {
  println("hello -> " + i)
}
```

13. While and Do while loop in scala?

```
// Main method
def main(args: Array[String])
{
    // variable declaration (assigning 5 to a)
    var a = 5;

    // loop execution
    do
    {
        println("a is : " + a);
        a = a - 1;
    }
    while (a > 0);
}
```

```
// variable declaration (assigning 5 to a)
var a = Array("do_while", "for", "while")
var index = 0

// loop execution
while (index < a.length)
{
    if(a(index) == "while")
        println("index of while is " + index)
    index = index + 1
}
```

14. Define List in Scala?

```
val mylist: List[Int] = List(1,2,5,8,9,6,4);
val names: List[String] = List("Max", "Tom", "John");
def main(args: Array[String]) {
  println(0 :: mylist);
  println(mylist);
  println(names);
  println(1 :: 5:: 9 :: Nil);
  println(mylist.head);
  println(names.tail);
  println(mylist.tail);
  println(names.isEmpty);
  println(mylist.reverse);
  println(List.fill(5)(2));
  println(mylist.max);

  mylist.foreach( println )
  var sum : Int = 0;
  mylist.foreach(sum += _);
  println(sum);

  for (name <- names) {
    println(name);
  }

  println(names(0));
```

```
<terminated> Demo$ [Scala Application] C:\Program File
List(0, 1, 2, 5, 8, 9, 6, 4)
List(1, 2, 5, 8, 9, 6, 4)
List(Max, Tom, John)
List(1, 5, 9)
1
List(Tom, John)
List(2, 5, 8, 9, 6, 4)
false
List(4, 6, 9, 8, 5, 2, 1)
List(2, 2, 2, 2, 2)
9
1
2
5
8
9
6
4
35
Max
Tom
John
```

15. How to define sets in Scala?

```
// Scala - Sets
object Demo {
  val myset: Set[Int] = Set(1,2,5,8,9,6,4);
  val myset2: Set[Int] = Set(4,2,9,18,19,16,14);
  val names: Set[String] = Set("Max", "Tom", "John")
  def main(args: Array[String]) {
    println(myset + 10);
    println(names("Maxxxx"));
    println(myset.head);
    println(myset.tail);
    println(myset.isEmpty);

    println(myset ++ myset2);
    println(myset.++(myset2))

    println(myset.&(myset2));
    println(myset.intersect(myset2));
    println(myset.max);

    println(myset.min);
```

```
<terminated> Demo$ [Scala Application] C:\Program Files\Java\jdk8\bin\javav
Set(5, 10, 1, 6, 9, 2, 8, 4)
false
5
Set(1, 6, 9, 2, 8, 4)
false
Set(5, 14, 1, 6, 9, 2, 18, 16, 8, 19, 4)
Set(5, 14, 1, 6, 9, 2, 18, 16, 8, 19, 4)
Set(9, 2, 4)
Set(9, 2, 4)
9
1
```

16. Map in scala?

```scala
/*   ======== Map  ========== */
val map1: Map[Int, String] = Map(1 -> "Ayushi", 2 -> "Rubal", 3 -> "xyz 123")

println(map1) /* print Map */
println(map1(1)) /* get value of '1' key */
println(map1.keys) /* get Map Keys */
println(map1.values) /* get Map values */
println(map1.isEmpty) /* check if Map is empty */
println(map1(50)) /* get value of invalid key will throw exception */
```

```
actice › main(args: Array[String])
scalaPractice ×
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ...
Map(1 -> Ayushi, 2 -> Rubal, 3 -> xyz 123)
Ayushi
Set(1, 2, 3)
MapLike.DefaultValuesIterable(Ayushi, Rubal, xyz 123)
false
Exception in thread "main" java.util.NoSuchElementException Create breakpoint : key not found: 50
    at scala.collection.immutable.Map$Map3.apply(Map.scala:325)
    at pack.scalaPractice$.main(scalaPractice.scala:81)
    at pack.scalaPractice.main(scalaPractice.scala)
```

17. Tuples in scala?

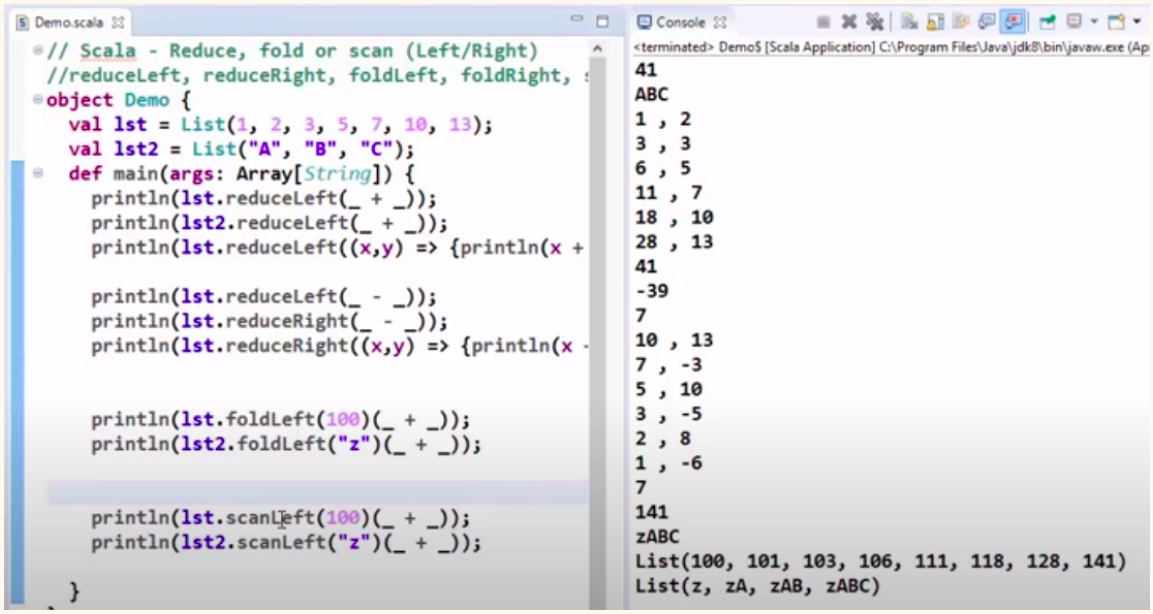
    Tuples can contain elements of different/heterogeneous data types. Tuples are immutable and they are fixed in size.

```scala
/*   ======== Tuples  ========== */
val tuple = (1, "hello", 12.50, 'a', false, (10, "world"))

/* Access tuple specific element */
println("Tuple 1st element- " + tuple._1)
println("Tuple 2nd element- " + tuple._2)
/* Get nested tuple element */
println("Tuple 6th element- " + tuple._6._1)

/* Iterate Tuple */
tuple.productIterator.foreach(x => println("element value- " + x))
```

There is another way to define Tuples using new keyword, depending on the size of the tuple elements use that Tuple, for example, if Tuple have 2 elements use Tuple2 -

```
/* Below tuple have 2 elements, so use "new Tuple2" */
val tuple1 = new Tuple2(1, "hello")

/* Below tuple have 5 elements, so use "new Tuple2" */
val tuple2 = new Tuple5(1, "hello", 12.50, 'a', false)
println(tuple1)
```

18. Reduce, fold and scan methods?



19. Scala classes and constructors?

Singleton class - If define a class with “object” keyword, it means the class is a singleton and can’t create an instance of this class.

```scala
package pack

/* This is singleton class, declare with "object" keyword */
object scalaPractice {

  def main(args: Array[String]): Unit = {
  }
}
```

Class -

```scala
/* Normal class without constructor */
class test1 {
}
```

20. Primary constructor and Auxiliary Constructors?

Primary constructor —>

```scala
/* class with primary constructor with private members */
class test2(private var id: Int, private val name: String) {
}

/* class with constructor when there is no keyword like val or var in-front of it
* then those members can't be access outside/inside of the class.
* Just initialize once creating object of this class, like --> new test3(12, "Ayushi") */
class test3(id: Int, name: String) {
}
```

Auxiliary constructor —>

- **Auxiliary** constructors are the alternative constructors for a class and are defined as methods in the class with the name "this".
- The auxiliary constructor in Scala is used for constructor overloading.
- A class can have many auxiliary constructors but they should have different signatures and must call any previously defined constructors.

```scala
/* class with Auxiliary constructor --> declare with this() method */
class test4(private var id: Int, private val name: String) {
  /* Auxiliary constructor with No Arguments */
  def this() {
    /* Call previous or primary constructor*/
    this(123, "Ayushi")
  }

  /* Auxiliary constructor with Arguments */
  def this(age: Int) {
    /* Call previous constructor*/
    this()
  }
}

object testObj {
  def main(args:Array[String]): Unit = {
    /* Initialize class with constructors */
    val obj1 = new test4()
    val obj2 = new test4( id = 123,  name = "Ayushi")
    val obj3 = new test4( age = 20)
  }
}
```

21. Inheritance in Scala?

    Scala doesn't support multiple inheritances just like java

    Super class

```
/* ====== Super class ========== */
class A {
  def printMethod(): Unit = {
    println("This is super class method")
  }
}

object testInheritance {
  def main(args: Array[String]): Unit = {
    val obj_B = new B()
    val obj_C = new C( text= "Hello World!!")

    obj_B.printMethod()
    obj_C.printMethod()
  }
}
```

Subclass - Use "override" keyword to override the parent method.

```
/* Subclass inherit parent class "A" */
class B extends A {
}
```

```
/* Subclass inherit parent class "A" and overriding parent class methods */
class C (val text:String) extends A {
  /* Override parent method */
  override def printMethod(): Unit = println("This is override in subclass-" + text)
}
```

22. Abstract class in Scala?

An abstract can be initialized. Abstract methods should be implemented in the subclass.

```
  val obj_1 = new A()
  val obj_B = new B
  val obj_C = new C
```

Class 'A' is abstract; cannot be instantiated

Super class

```scala
/* ====== Super Abstract class =========== */
abstract class A {
  def printMethod(): Unit = {
    println("This is super class method")
  }

  /* 1st abstract method */
  def abstractMethod(): Unit

  /* 2nd abstract method */
  def abstractMethod2(): String
}
```

Subclass -

```scala
/* Subclass inherit parent class "A" */
class B extends A {
  override def abstractMethod(): Unit = ???

  override def abstractMethod2(): String = ???
}
```

23. Scala lazy Evolution?

    Scala supports strict evolution and lazy evolution

```scala
//Scala: Lazy Evaluation

class strict {
  val e = {
    println("strict");
    9
  }
}
class LazyEval {
  lazy val l = {
    println("lazy");
    9
  }
}

object Demo {
  def main(args: Array[String]) {
     val x = new strict;
     val y = new LazyEval;

     println(x.e);
     println(y.l);

  }
}
```

```scala
//Scala: Lazy Evaluation



object Demo {
  def method1(n: Int) {
    println("Method 1");
    println(n);
  }
  def method2(n: => Int) {
    println("Method 2");
    println(n);
  }

  def main(args: Array[String]) {
    val add = (a : Int, b: Int) => {
      println("Add");
      a + b
    }

    method1(add (5,6));

    method2(add (5,6));

  }
}
```

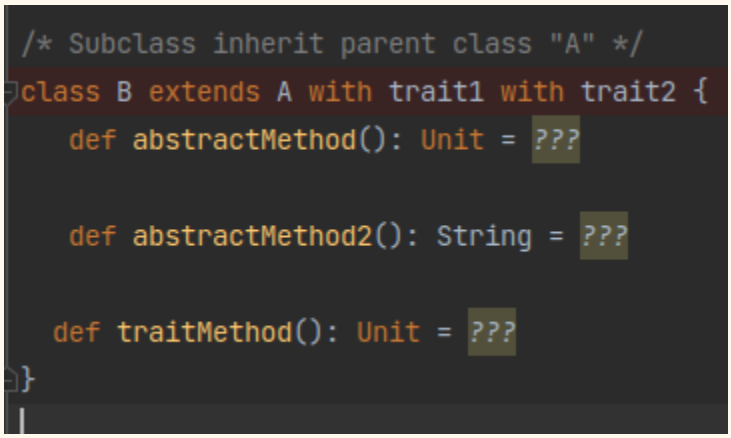
```
<terminated> Der
Add
Method 1
11
Method 2
Add
11
```

24. Scala Trait?

Traits are like interfaces, they are partially implemented interfaces. We can inherit Trait using the “with” keyword just like “implements” keyword in java.

```scala
/* Trait example, it can have both type of methods - with definition or without definition */
trait trait1 {
  def traitPrintMethod(): Unit = {
    println("This is super class method")
  }

  /* without definition method */
  def traitMethod(): Unit
}

trait trait2 {
  def trait2PrintMethod(): Unit = {
    println("This is super class method")
  }
}
```

Just add “with” to inherit as many Traits you want, just like below we have added “with” two times to inheriting each Trait.



25. What is function currying in Scala?

Currying in Scala is simply a technique or a process of transforming a function. This function takes *multiple arguments* into a function that takes single argument.

```scala
// Scala program add two numbers
// using currying Function
object Curry
{
    // Define currying function
    def add(x: Int, y: Int) = x + y;

    def main(args: Array[String])
    {
        println(add(20, 19));
    }
}
```

```scala
object Curry
{
    // transforming the function that
    // takes two(multiple) arguments into
    // a function that takes one(single) argument.
    def add2(a: Int) = (b: Int) => a + b;

    // Main method
    def main(args: Array[String])
    {
        println(add2(20)(19));
    }
}
```

26. Partially used functions in scala?

    When a function is not able to produce a return for every single variable input data given to it then that function is termed as **Partial function**.

27.

PySpark

```
===============
dataread
===============
data = sc.textFile("file:///C://data//txns_head")

===============
```

```
Filter data
===============
filterdata= data.filter(lambda x: 'Gymnastics' in x)

data = sc.textFile("file:///home/cloudera/data/txns")
flatdata= data.flatMap(lambda x:x.split(","))

==================================
Create Schema Rdd using case class
==================================
from collections import namedtuple
schema = namedtuple("schema", ["txnno",
"txndate","custid","amount","category","product","city","state","spendby"])

gymdata=data.filter(lambda x: 'Gymnastics' in x)
mapsplit=gymdata.map(lambda x:x.split(","))
schemardd=mapsplit.map(lambda
x:schema(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
schemafilter=schemardd.filter(lambda x: 'Gymnastics' in x.product)

===============
Create Row Rdd with Struct
===============
from pyspark.sql import Row
from pyspark.sql import *

data = sc.textFile("file:///C://data//txns_head")
mapdata= data.map(lambda x:x.split(","))
rowdata = mapdata.map(lambda
x:Row(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
filterdata.foreach(print)

===============
Schema rdd to Dataframe
===============
df = schemafilter.toDF()

===============
Row rdd to Dataframe
===============
from pyspark.sql.types import *

strct = StructType([ \
    StructField("txnno",StringType(),True), \
```

```
    StructField("txndate",StringType(),True), \
    StructField("custid",StringType(),True), \
    StructField("amount", StringType(), True), \
    StructField("category", StringType(), True), \
    StructField("product", StringType(), True), \
    StructField("city", StringType(), True), \
    StructField("state", StringType(), True), \
    StructField("spendby", StringType(), True)\
  ])

data = sc.textFile("file:///C://data//txns_head")
mapdata= data.map(lambda x:x.split(","))
rowdata = mapdata.map(lambda
x:Row(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
df = spark.createDataFrame(rowdata,strct)
df.printSchema()
df.show()

=======================================
csv read to parquet write seamless
=======================================
df =
spark.read.format("csv").option("header","true").load("file:///C://data//tx
ns_head")
df.write.format("parquet").save("file:///C://data//pysparkparquet_data")
print("written done")
```

Read data from URL

```
==================================
import urllib
from urllib import request

fp = urllib.request.urlopen("https://randomuser.me/api/0.8/?results=10")
mybytes = fp.read()
mystr = mybytes.decode("utf8")
print(mystr)

rdd = sc.parallelize([mystr])
df = spark.read.json(rdd)

from pyspark.sql.functions import *
```

```
df.withColumn("results",explode(col("results")))
.select("nationality","seed","version","results.user.username","results.user.cell","results.user.dob","results.user.email","results.user.gender","results.user.location.city","results.user.location.state","results.user.location.street","results.user.location.zip","results.user.md5","results.user.name.first","results.user.name.last","results.user.name.title","results.user.password","results.user.phone","results.user.picture.large","results.user.picture.medium","results.user.picture.thumbnail","results.user.registered","results.user.salt","results.user.sha1","results.user.sha256")
.show()
```

Spark hive

```
df1.write.format("parquet").saveAsTable("test.pysparkparquet")
```

Spark Deployment

```
spark-submit --master local[*] --conf
"spark.driver.extraClassPath=/home/cloudera/avrojar/*" Pysparkdeploy.py
```

Pyspark Add extra jars

```
============================
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars C:/avrojar/*pyspark-shell'
spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext

df =
spark.read.format("csv").option("header","true").load("file:///C://data//txns_head")
df.write.format("com.databricks.spark.avro").mode("overwrite").save("file://C://data//avrowrite")
print("avro written")
```

Phase 2 - Project

Requirements ⇒

1. Create a project in Eclipse and do the necessary configuration
2. Add avro jar and Read sample avro data
3. Read URL data and convert to Dataframe https://randomuser.me/api/0.8/?results=500
4. Flatten the URL data (Dont select those Capital Letter Columns)
5. Remove Numericals from username from flattened data
6. Do the left broadcast Join
    a. AVRODATA ⇒ Broadcast Join with (Numericals removed URL data) using 'username' Column
7. Create two dataframes after the Join with filter Nationality null and Nationality Not Nullm assign name them like following ⇒
    i. available_Customers (filter Nationality Not Null)
    ii. not_Available_Customers (filter Nationality Null)
8. For **not_Available_Customers** dataframe replace all the String column Null to "**NOT AVAILABLE**" and non-string Columns to **0**
9. Take both available_customers and **not_Available_Customers dataframe** ⇒ Attach current_date column at the end of table

```
package pack

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import org.apache.spark.{SparkConf, SparkContext}

import scala.io.Source


object projectObj {

 def main(arg: Array[String]): Unit = {

   val conf = new SparkConf().setAppName("Spark Project").setMaster("local[*]")
   val sc = new SparkContext(conf)
   val spark = SparkSession.builder().getOrCreate()
   sc.setLogLevel("error")
```

```scala
/* Read AVro file */
val avroDF = spark.read.format("avro").load("file:///home/Practice/data/projectsample.avro")
avroDF.show(false)

/* Read data from API */
var apiData = Source.fromURL("https://randomuser.me/api/0.8/?results=200")
              .mkString
println(apiData)

/* convert 'apiData' to dataframe */
val rdd = sc.parallelize(List(apiData))
val apiDF = spark.read.json(rdd)
apiDF.show(10)
apiDF.printSchema()

/* flatten(process) complex api json data */
var flattenApiDF = apiDF.withColumn("results", explode(col("results")))
  .withColumn("cell", col("results.user.cell"))
  .withColumn("dob", col("results.user.dob"))
  .withColumn("email", col("results.user.email"))
  .withColumn("gender", col("results.user.gender"))
  .withColumn("city", col("results.user.location.city"))
  .withColumn("state", col("results.user.location.state"))
  .withColumn("street", col("results.user.location.street"))
  .withColumn("zip", col("results.user.location.zip"))
  .withColumn("md5", col("results.user.md5"))
  .withColumn("first_name", col("results.user.name.first"))
  .withColumn("last_name", col("results.user.name.last"))
  .withColumn("title", col("results.user.name.title"))
  .withColumn("password", col("results.user.password"))
  .withColumn("phone", col("results.user.phone"))
  .withColumn("large_picture", col("results.user.picture.large"))
  .withColumn("medium_picture", col("results.user.picture.medium"))
  .withColumn("thumbnail_picture", col("results.user.picture.thumbnail"))
  .withColumn("registered", col("results.user.registered"))
  .withColumn("salt", col("results.user.salt"))
  .withColumn("sha1", col("results.user.sha1"))
  .withColumn("sha256", col("results.user.sha256"))
  .withColumn("username", col("results.user.username"))
  .drop("results")

flattenApiDF.show(5, false)

/* Remove Numerical from username from flattened api data */
```

```
   flattenApiDF = flattenApiDF.withColumn("username", regexp_replace(col("username"), "(\\d+)", ""))
   // another option to replace numeric in column
   //    flattenApiDF = flattenApiDF.withColumn("username", regexp_replace(col("username"), "([0-9])", ""))
   flattenApiDF.show(5, false)

   /* broadcasting on avroDF and performing left join */
   val joinDF = avroDF.join(broadcast(flattenApiDF), Seq("username"), "left")
   joinDF.show()

   /* Create two dataframes after the Join with filter Nationality 'null' and 'Not Null' */
   val not_available_Customers = joinDF.filter(col("nationality").isNull)
   val available_Customers = joinDF.filter(col("nationality").isNotNull)
   not_available_Customers.show()
   available_Customers.show()

   /* For not_available_Customers replace all the null column(String) to "NOT AVAILABLE" and Non String
Columns => 0 */
   val not_available_Customers2 = not_available_Customers.na.fill(0).na.fill("NOT AVAILABLE")

   /* Add current_date column at the end of table */
   available_Customers.withColumn("current_date", current_date()).show()
   not_available_Customers2.withColumn("current_date", current_date()).show()
 }
}
```

Agile Methodology

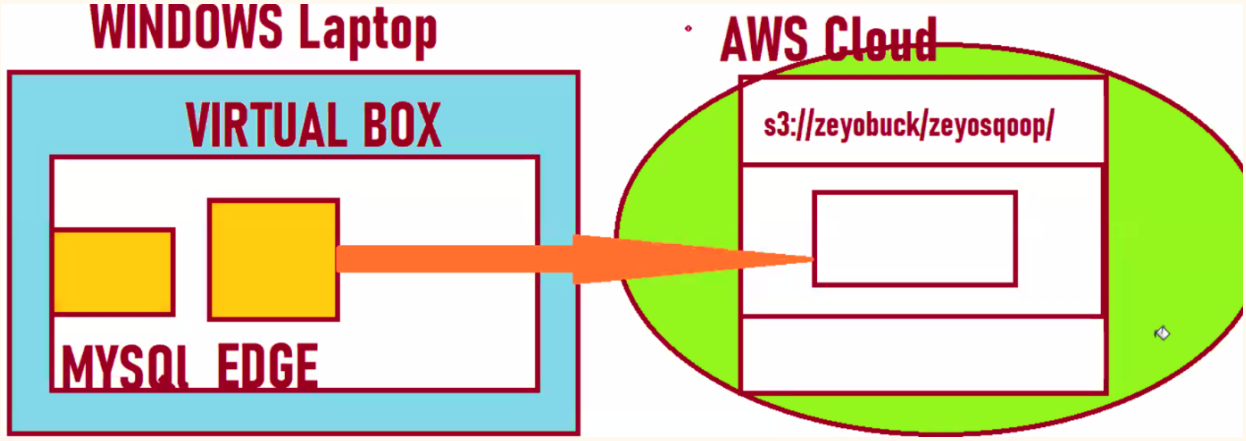
What is your day to day activity?

- We follow agile Methodology
- We have 2 weeks of Sprint
- Scrum master assigns stories for us.
- We have to progress that story every day!
- If I could not complete that story within 2 weeks It spilled to the next sprint (which is not recommended)
- We have daily scrum call to discuss progress on stories and blockers
- We have sprint retros also and we have JIRA board

AWS

Cloud is the usage of services. Few are the below services -

- **EC2** (Elastic compute cloud) - It is for getting a virtual laptop
- **RDS** (Relational database system) - It is for storing RDBMS data
- **S3** (Simple Storage service/Scalable Storage) - It is storage to store any data.
- **EMR**(Elastic Map Reduce) - For rent Hadoop system, EMR is a group of nodes with hadoop installed in it.



BOSS --- Sai You have to do a course certification
You need to write an exam but to write an exam you need windows SSD laptop

But sai does not have it he has very basic laptop. lets rent a laptop. Sai is searching is there any one can give WINDOWS SSD laptops for Rental

Sai found three ways to rent Laptop -

1. **AWS - *JEFF***
2. Google Cloud Platform - Sunder
3. Azure - Satya Nadela

Sai Decided to go with Jeff.

Sai --- Hey Jeff can you courier the Laptop

Jeff ---- No No no I will not do that . I wil have the SSD laptop with me -- you have to connect remotely

Sai ----- Sure I will do it But how can I connect to that Laptop

Jeff ---- You have webportal just like Flipkart,Amazon - i have my own portal for rental business. Create your own account and add windows laptop to Cart page and check it out
I assure you, your laptop will be available in just 2 minutes.
Once done. Pay me for the minutes you used.

Sai --- whats you WEBPAGE

Jeff --- My rental Product name is Amazon Web Services - AWS

Sai ---- I will create my own account and reach you out. Hey I have create my own account.Next steps

Jeff --- Login first

Sai --- Sure ,i have selected Windows Laptop with SSD and launched its in Running state State

Jeff ---- Yes that mean you laptop is ready to use but wait for 2 Minutes ---- Steps to Connect lllllllllllllllll

Sai ---- Thanks jeff I completed my work. I will shut it down I used for 2 hours

Jeff ---- Sure just pay me 30rs

Sai --- Dear Jeff. I like your service
But I need another Help -- I own zeyobron.
My students was mysql for practise. Can you help me

JEFF --- Dont worry,I have service known RDS.You can go ahead and create MYSQL server

Configure AWS

- **For Windows install the below application** -

  AWS CLI → https://awscli.amazonaws.com/AWSCLIV2.msi

  GIT download → https://github.com/git-for-windows/git/releases/download/v2.38.0.windows.1/Git-2.38.0-64-bit.exe

- **For Linux**

  Install AWS CLI - (Run below commands)

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
```

```
sudo ./aws/install
```

- Then run `aws configure` and provide access, secret and region properly.

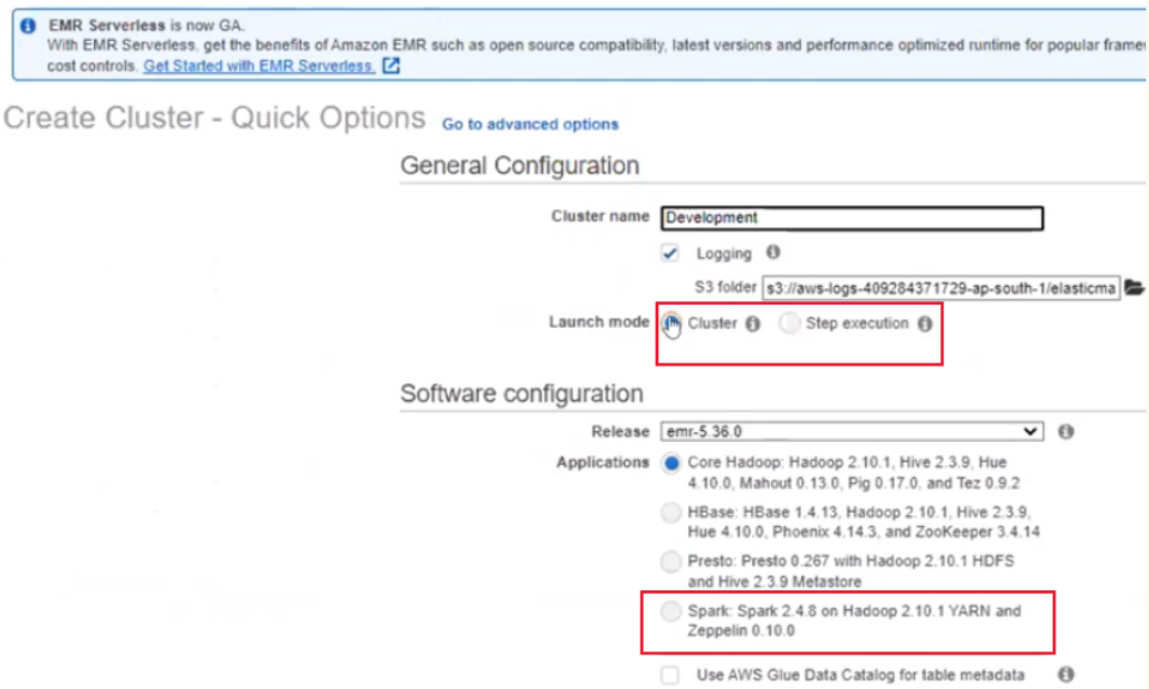
Create Bucket and files in AWS using terminal

```
aws s3 mb s3://<name>newbuck // create bucket
aws s3 ls        // list bucket
echo zeyobron>zeyo.txt      // create file
aws s3 cp zeyo.txt s3://<name>newbuck // copy file in the bucket
aws s3 ls s3://<name>newbuck/     // list bucket
aws s3 rm s3://<name>newbuck/zeyo.txt     // remove file from bucket
aws s3 rb s3://<name>newbuck/    // remove bucket
```

S3-dist-cp --src /user/hadoop/srcFolder --desc s3://com.zeyo.dev/descFolder

Create cluster and Develop code here

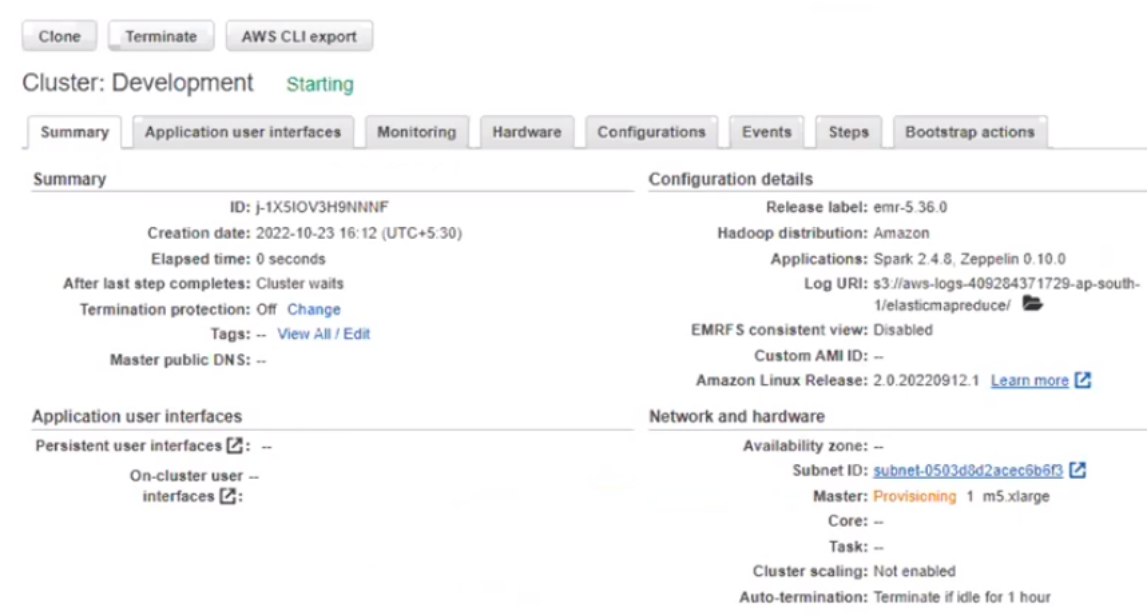
1. Got to EMR and Create cluster using necessary configuration -



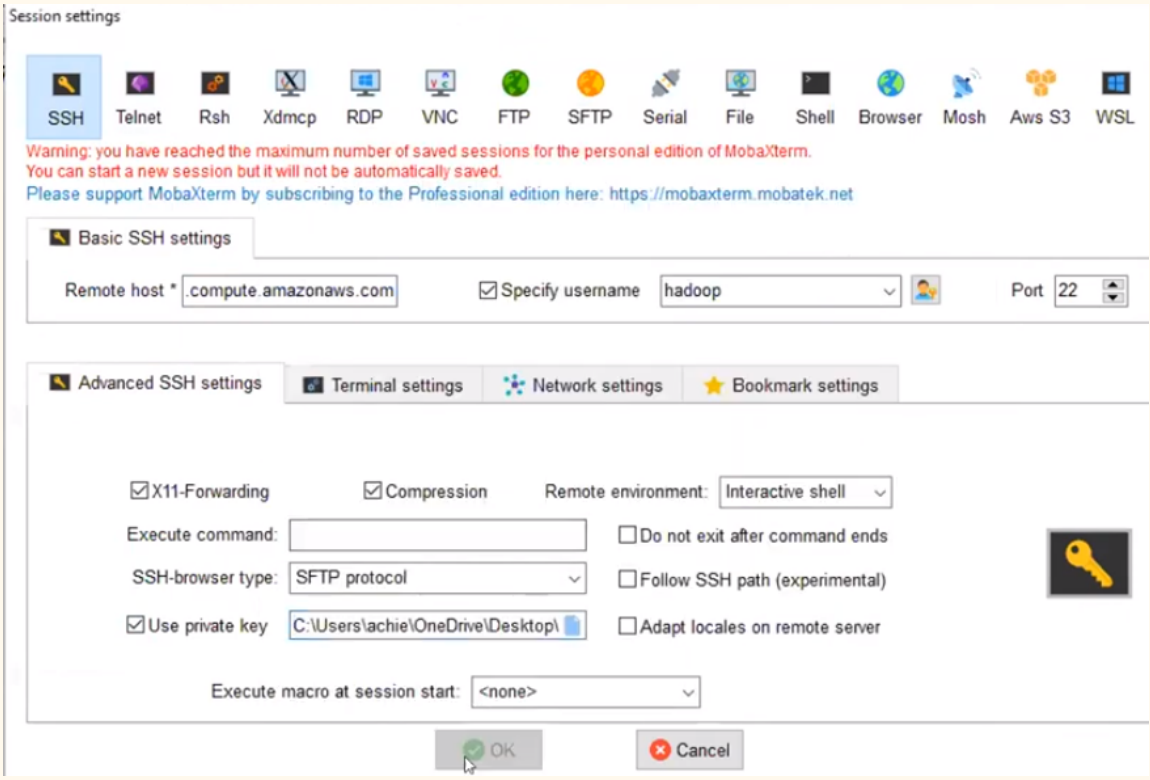
- For **development** of code select **Cluster** mode
- For **deployment** select **Step execution** mode

2. Then provide Hadoop configuration, like no. of nodes, Then click on " **Create Cluster**"

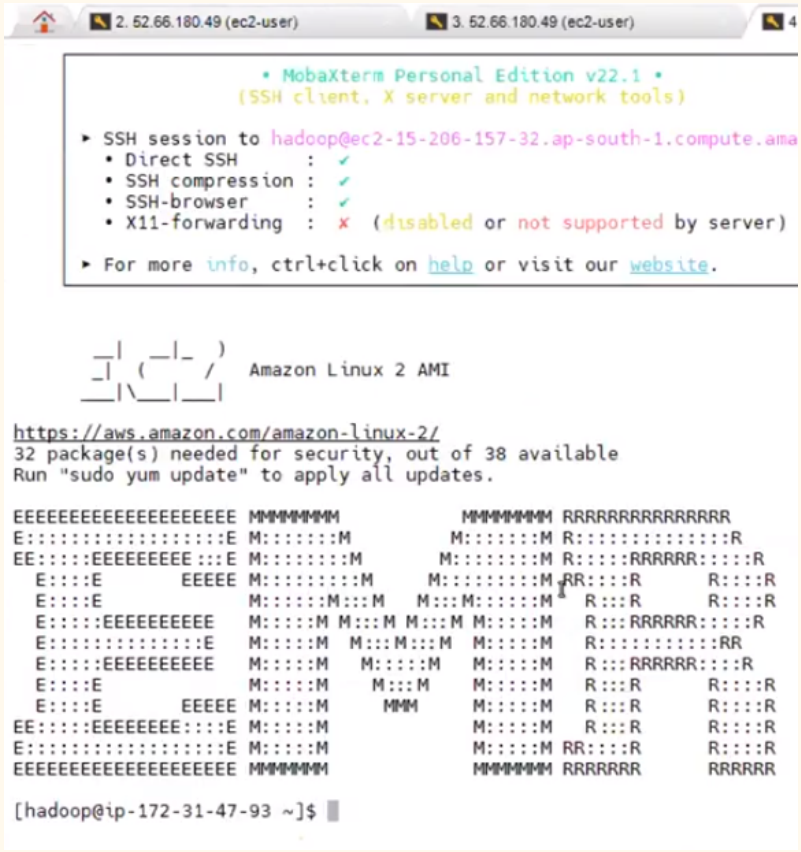
3. Then cluster will **start =>**



4. Login to cluster using Putty or Mobextram and give cluster's private key(PPM) ⇒



5. You should be able to login in cluster -



6. Then Run command **spark-shell** to develope code in scala.
7. If development code gonna need extra jar like ‘avro’ or ‘mysql-connector’, install jar using ⇒

```
spark-shell --package org.apache.spark:spark-avro.2_11:2.4.6
```

8. Then write or copy paste your code from IDE to spark-shell ⇒
   Either use **:paste** mode and copy paste all multiline code here or paste you code one by one

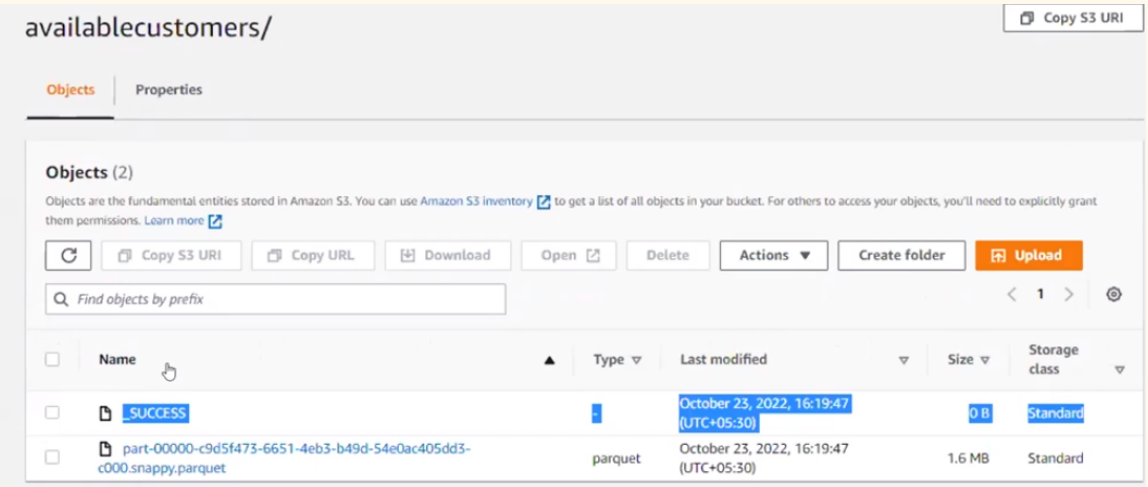
```
scala> :paste
// Entering paste mode (ctrl-D to finish)

import org.apache.spark.broadcast._
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions.broadcast
import java.time.{ZonedDateTime, ZoneId}
import java.time.format.DateTimeFormatter
import scala.io.Source

val data = spark.read.format("avro")
                              .load("s3://com.zeyo.analytics.dev/srcfolder/")
                              data.show()

                              val html = Source.fromURL("https://randomuser.me/api/0.8/?results=1000")
                              val s = html.mkString
```

```
scala> notnull_with_current_date.show()
22/10/23 10:48:38 WARN TaskSetManager: St
task size is 100 KB.
```

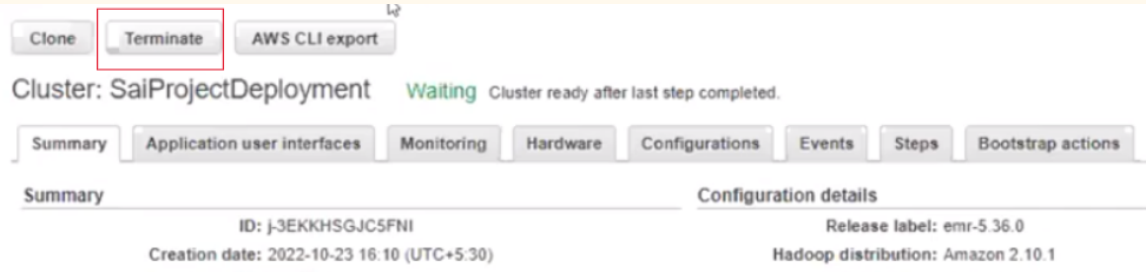
9. Then write processed data in destination location like S3 or Hive anywhere as per requirement

```
scala> replacenull_with_current_date.write.format("parquet").mode("overwrite").save("s3://com.zeyo.analytics.dev/targe
tfolder/saidir/notavailablecustomers")
```

10. Chek in destination location either data has been saved there or not. Once data is written development is completed and Terminate cluster.



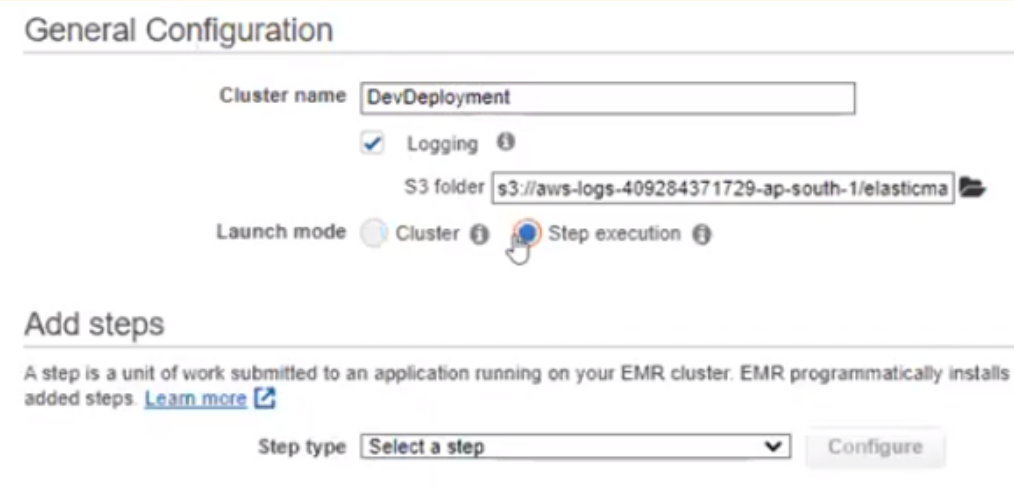
11. Then Terminate the cluster ⇒



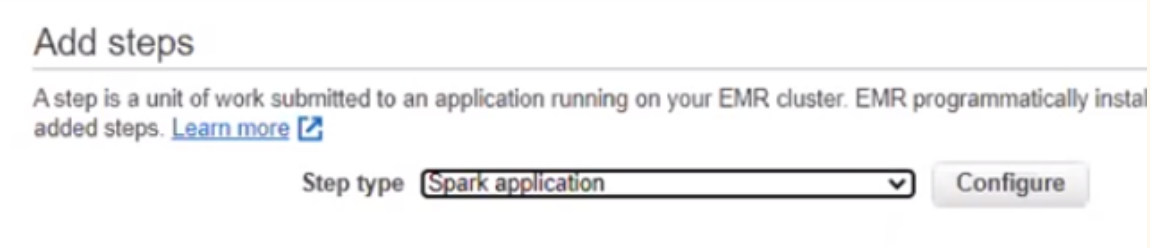
Create Cluster EMR Step execution(Deployment part)

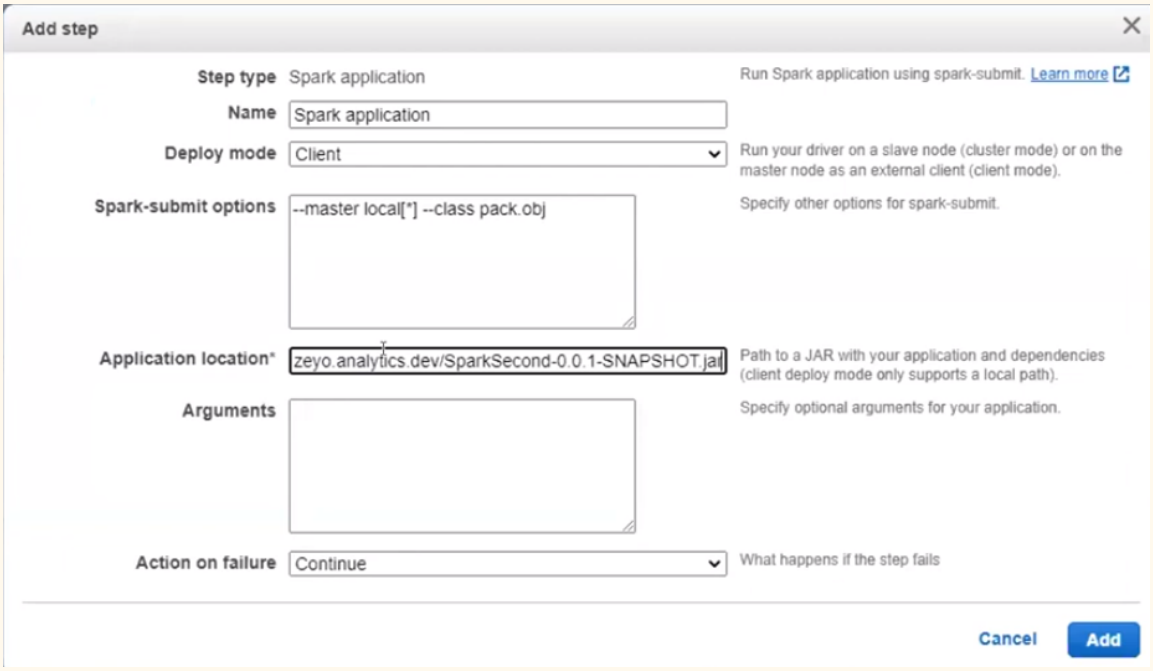
```
Create a cluster develop your Code
Put the code to eclipse and generate the Jar
Copy that Jar to s3
Create Step Execution Script using Template
Schedule that Script
Every the job runs on a time
```

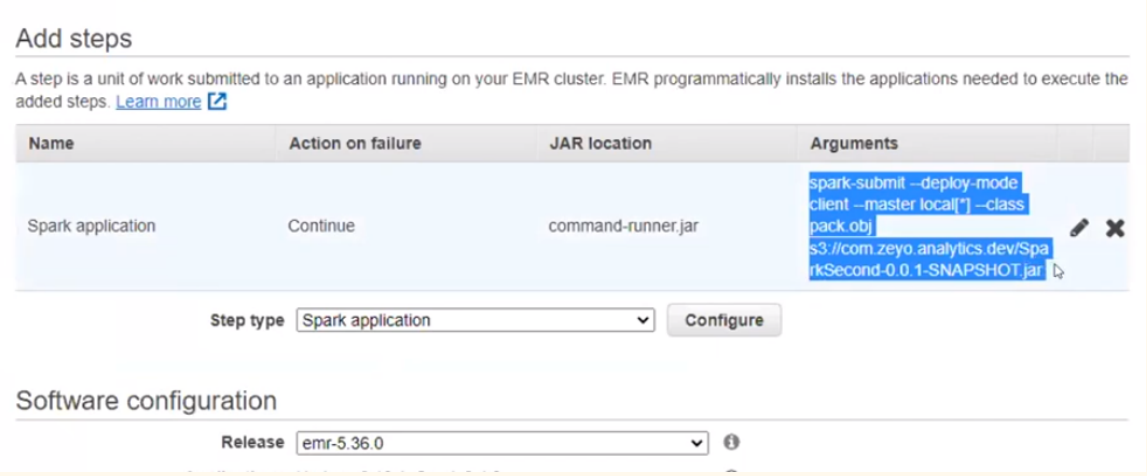
1. Create Jar of code ⇒ code which you developed in spark-shell, copy paste it in any IDE and create jar of it.
2. Copy the jar in S3 location under any bucket.
3. Give cluster name, select **Step execution** mode and other necessary configuration -



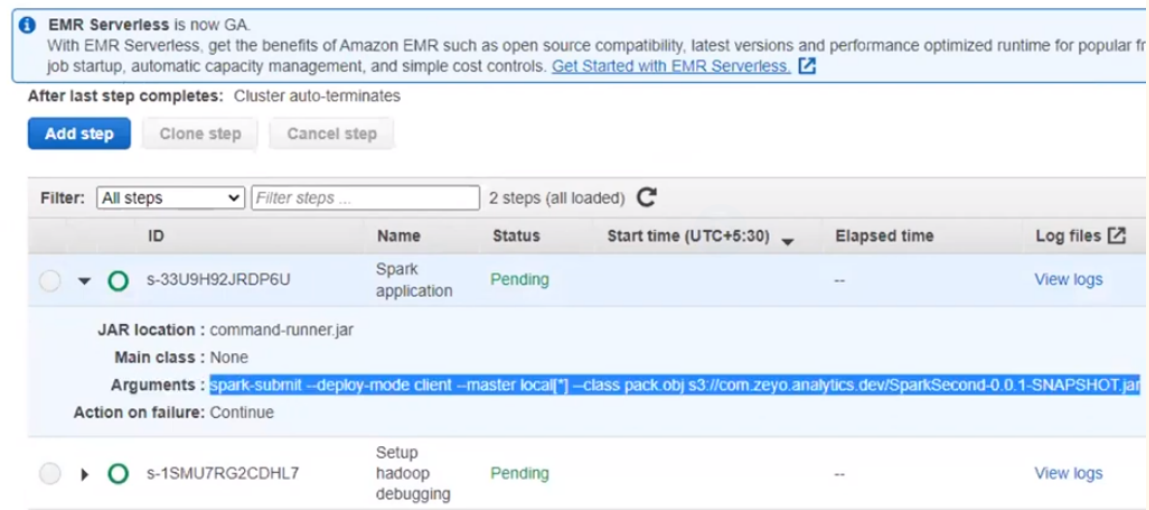
4. Create a cluster using **Step execution procedure** ⇒ select step type and click on configure ⇒ then provide the below details -



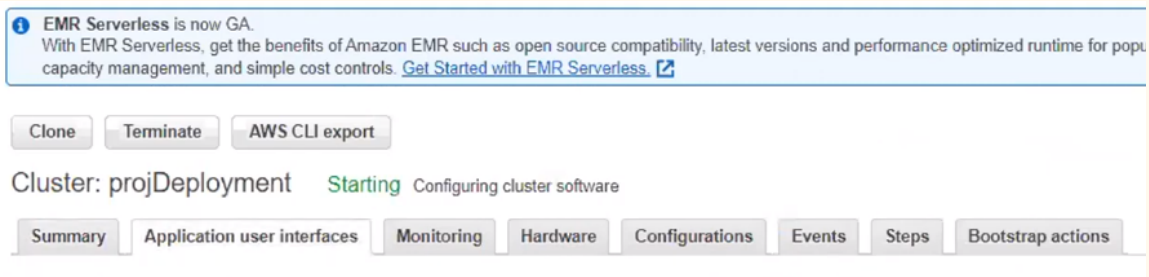




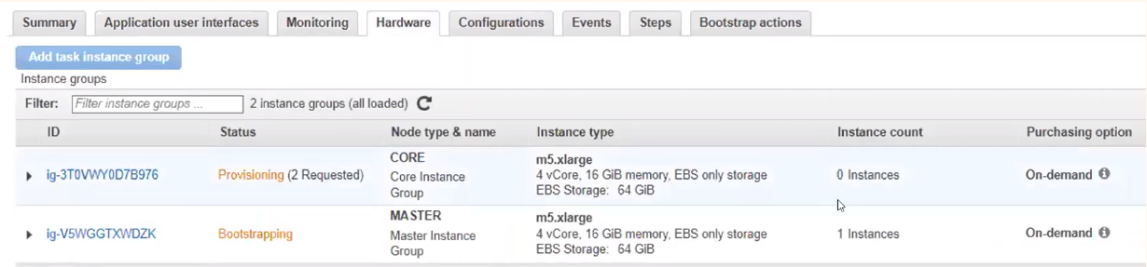
5. Then provide Hadoop configuration, like no. of nodes, Then click on " **Create Cluster**".
6. Then cluster will **start ⇒ then Steps will Run ⇒ Then cluster will terminate**
7. Then click on 'Steps' where you will see your steps are executing in cluster



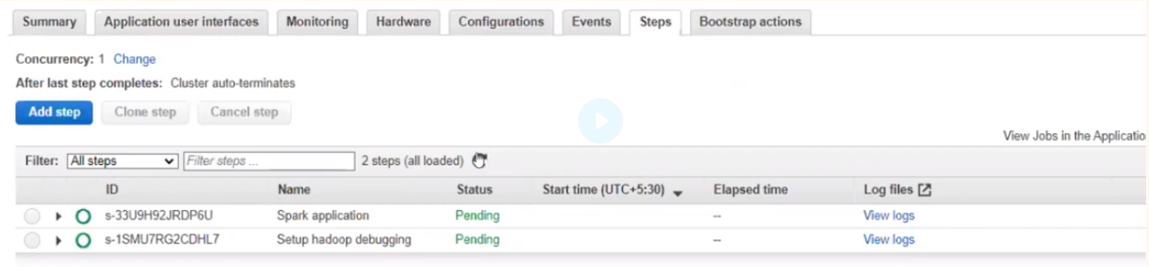
8. Once all steps are completed, cluster will be **Terminated automatically.**
9. In cluster we can see application ids and and check their logs -



10. Go to **hardware** option and check how many nodes are there - for example below we have 2 slave nodes and 1 is master node



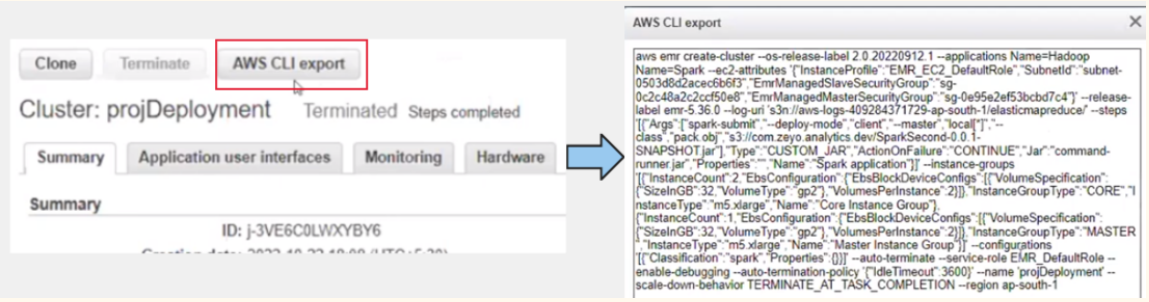
11. Check how many steps are executed -



12. Once cluster is completed then all processed data will be saved in destination location as mention in develop code.

How to automate AWS execution

1. Take any cluster job and click on AWS CLI report

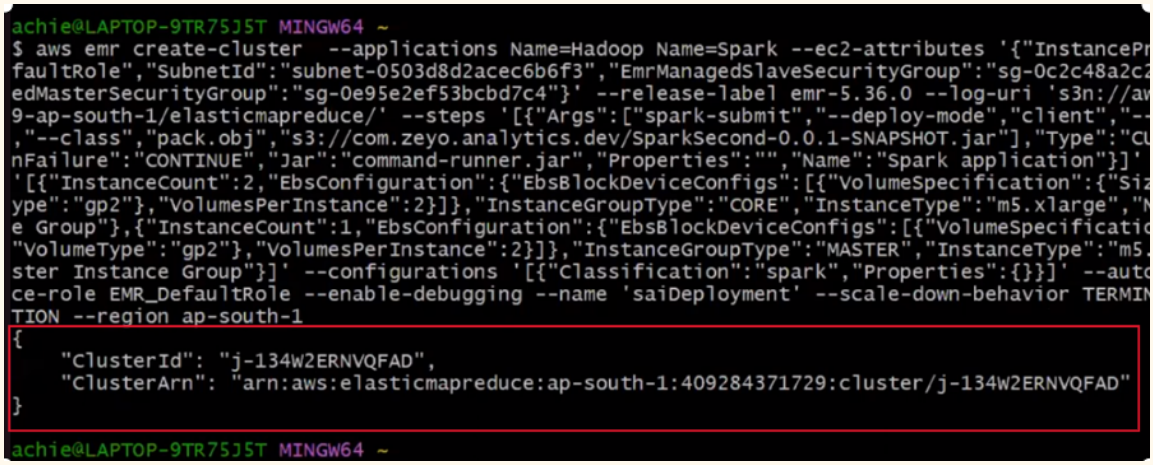


2. Copy the command and edit as per the new job configuration in notepad.

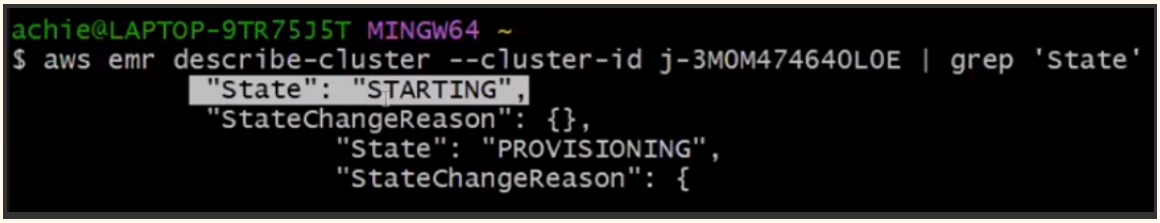
   To add maven library like Avro, ad jar using `--package org.apache.spark:spark-avro.2_11:2.4.6`

```
aws emr create-cluster  --applications Name=Hadoop Name=Spark --ec2-attributes
'{"InstanceProfile":"EMR_EC2_DefaultRole","SubnetId":"subnet-0503d8d2acec6b6f3","EmrManagedSlaveSecurityGroup
":"sg-0c2c48a2c2ccf50e8","EmrManagedMasterSecurityGroup":"sg-0e95e2ef53bcbd7c4"}' --release-label emr-5.36.0
--log-uri 's3n://aws-logs-409284371729-ap-south-1/elasticmapreduce/' --steps
'[{"Args":["spark-submit","--deploy-mode","client","--master","local[*]","--class","pack.obj","s3://com.zeyo.
analytics.dev/SparkSecond-0.0.1-SNAPSHOT.jar"],"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE","Jar":"comman
d-runner.jar","Properties":"","Name":"Spark application"}]' --instance-groups
'[{"InstanceCount":2,"EbsConfiguration":{"EbsBlockDeviceConfigs":[{"VolumeSpecification":{"SizeInGB":32,"Volu
meType":"gp2"},"VolumesPerInstance":2}]},"InstanceGroupType":"CORE","InstanceType":"m5.xlarge","Name":"Core
Instance
Group"},{"InstanceCount":1,"EbsConfiguration":{"EbsBlockDeviceConfigs":[{"VolumeSpecification":{"SizeInGB":32,
"VolumeType":"gp2"},"VolumesPerInstance":2}]},"InstanceGroupType":"MASTER","InstanceType":"m5.xlarge","Name":
"Master Instance Group"}]' --configurations '[{"Classification":"spark","Properties":{}}]' --auto-terminate
--service-role EMR_DefaultRole --enable-debugging --name 'saiDeployment' --scale-down-behavior
TERMINATE_AT_TASK_COMPLETION --region ap-south-1
```

3. After the configuration of the above command, run it in the terminal. Then cluster will created and the processed data will also store in S3 and the cluster will be terminated. Also after running the command one cluster id will be generated -



4. Then check the status of the cluster in the terminal - `aws emr describe-cluster --cluster-id <CLUSTERID> | grep 'State'`

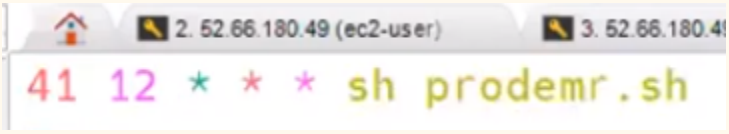


5. Run the command to check whether the destination folder is created in S3 or not - `aws s3 ls s3://com.zeyo.analytics.dev/destfolder/`

How to do the deployment in Production

1. Copy jar in AWS S3 location (dev env bucket).
2. Save the spark-submit command in .sh file, like ⇒ **emr.sh** , and update this .sh file in the same s3 location where the jar is located.
3. Then cloudops team will create production **EC2** and after login copy the jar file from dev S3 bucket to prod S3 bucket
4. Copy **emr.sh** file in production EC2 locally and update sh file as per production.
5. Run this **sh** file in production Ec2 using the command ⇒ `sh emr.sh` then the prod cluster will start to run.

6. **How cloudops team schedules this spark-submit procedure** - They use tool name **nifi** or **contab**
7. Run `crontab -e` in the same production EC2 and schedule spark-submit using sh file name, like below



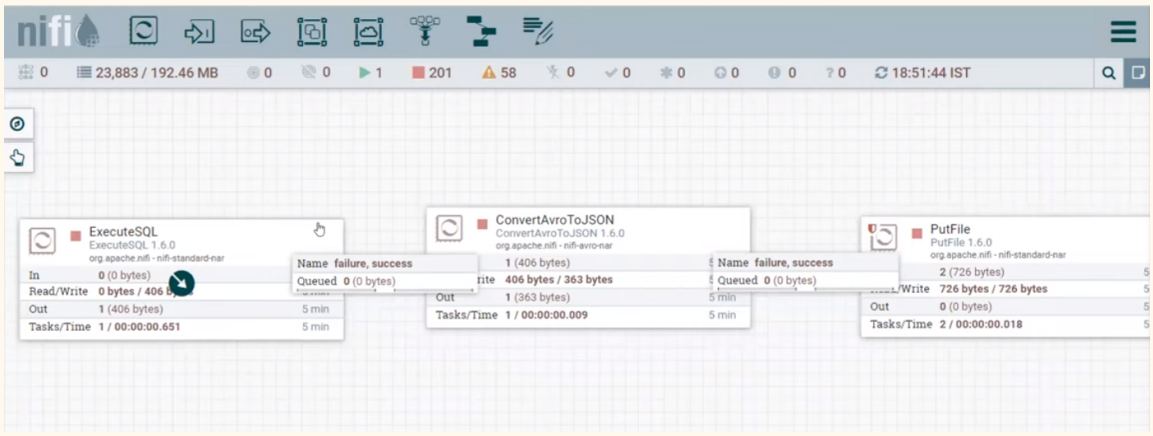
Nifi

Nifi is an **orchestration** tool. Nifi can be used for scheduling, automation process, and Ingestion tool.

For **orchestration,** there are other tools that can be used -

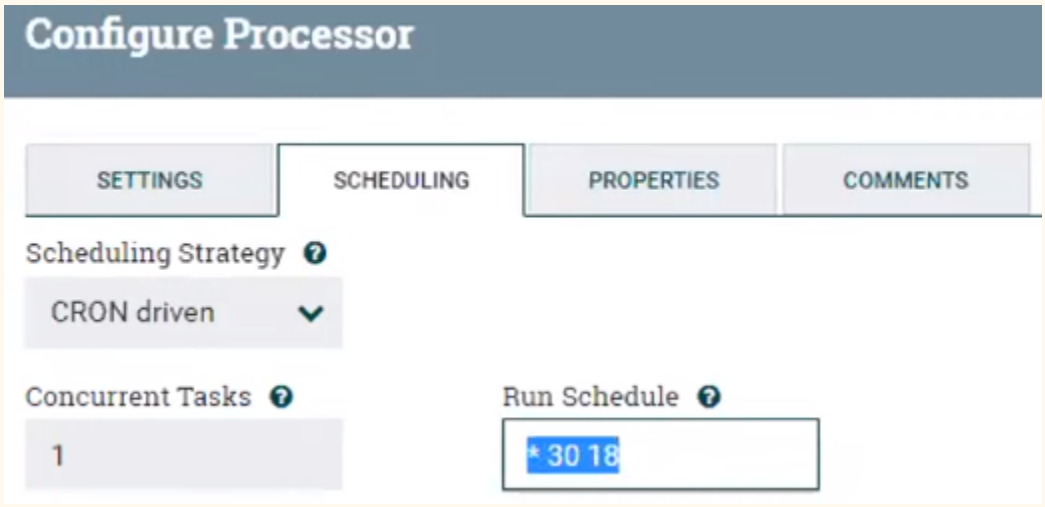
- Oozi - AWS
- Airflow - Azure
- Talend

Nifi version - 1.6.0

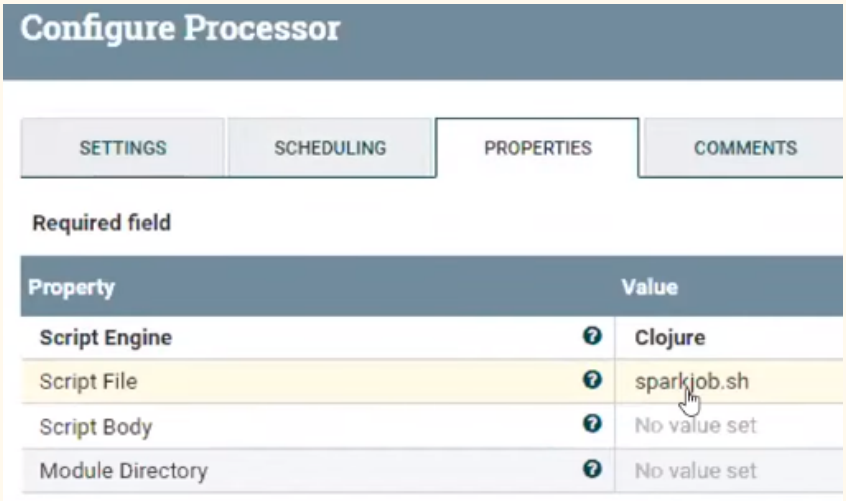


Schedule job using Nifi

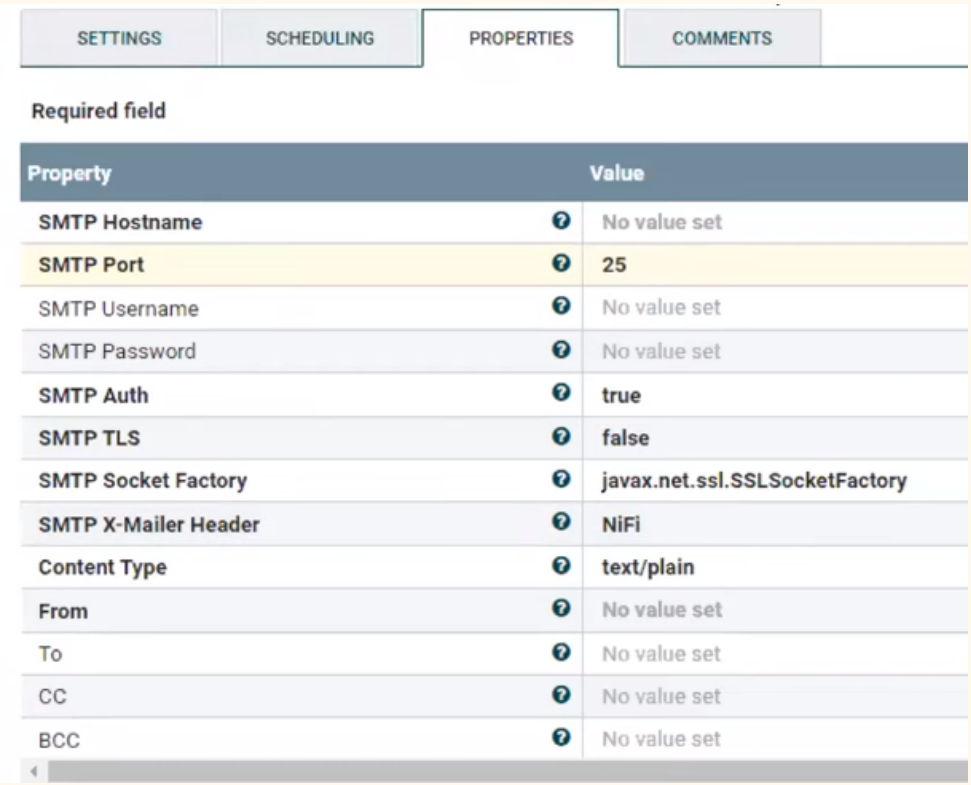
1. Create **execute script module** and Give Schedule time



2. Give sh file



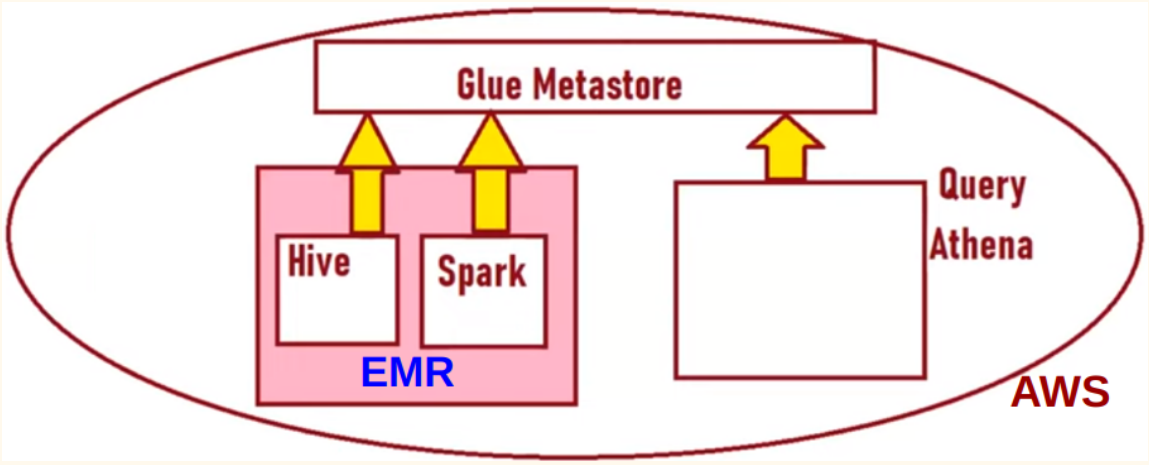
3. Create a module for **putEmail** once the script is completed, under **SMTP Hostname** give email address



AWS Athena

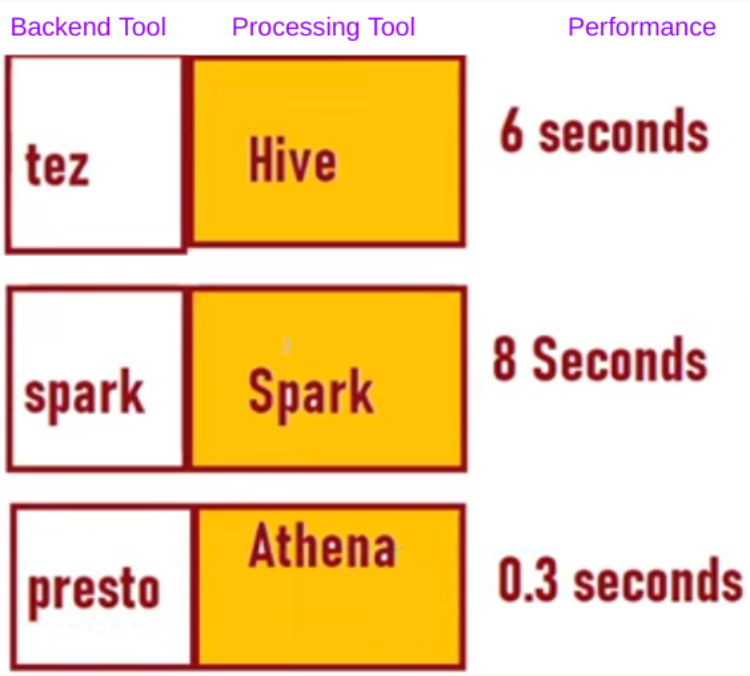
- Athena is an interactive query service.
- Athena analyzed data in S3 using SQL query.
- Athena can process structured, unstructured and semi-structured data.
- Athena used Presto engine for performance.
- Athena is an analytics tool, it is same as hive.
- As Hive runs on the top of HDFS, the same Athena runs on the top of S3.

- When we create EMR all hive, spark, and Athena points to the same Metastore(**AWS Glue**) -
  AWS Glue have meta information for all tables/data.



- **Athena performance is much more faster than Hive and Spark.**

  Hive uses Tez engine in EMR.

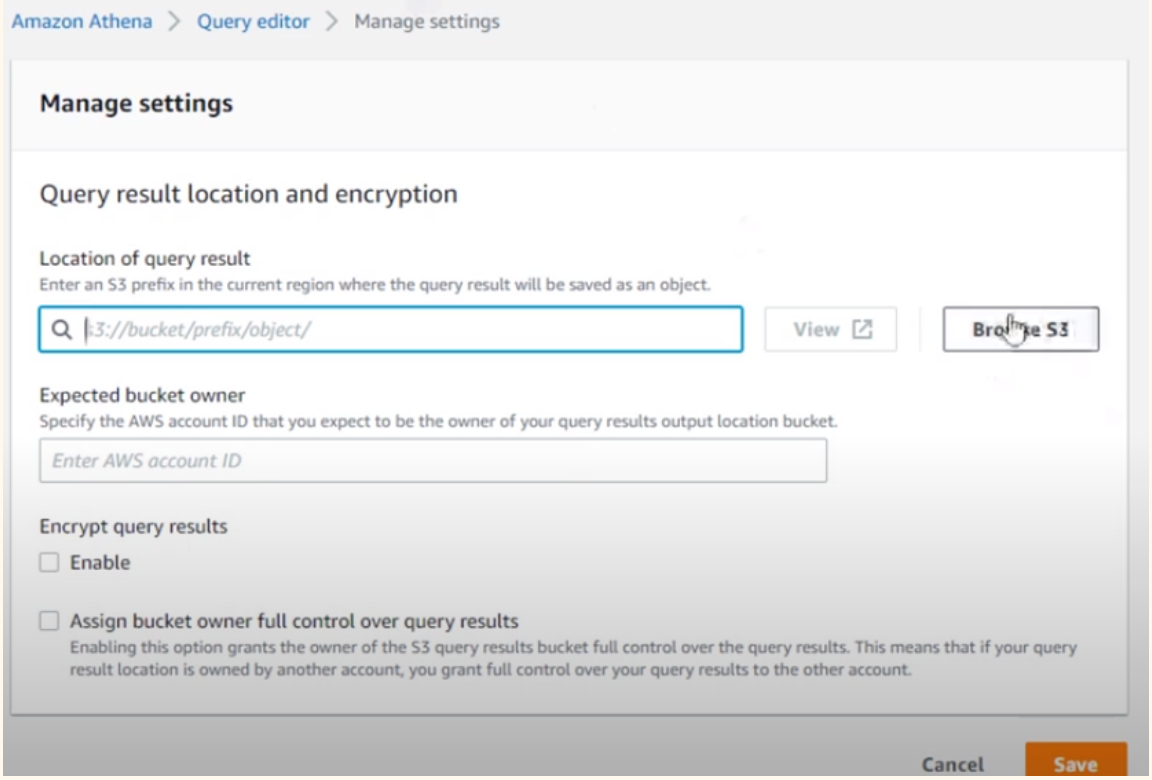


Ques - If we terminate EMR what will happen,will we able to query table from athena?

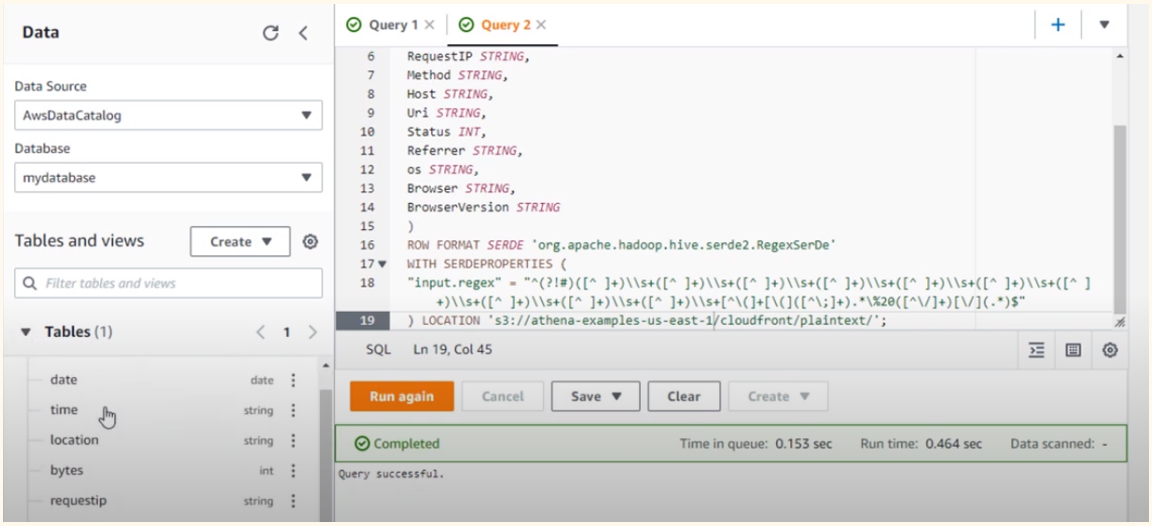
Ans - Data exist in S3 but what about table. Here comes AWS Glue where table metastore/schema is created, so even if EMR is terminated we have data in S3 and table schema/metastore exist in Glue, we can query tables in Athena. But to create metastore in Glue we have to enable below checkbox while creating EMR -

Start Athena

1. Create bucket in S3 to store query results it is the same as Hive a=have its warehouse in HDFS
2. Then Go to Athena ⇒ Setting ⇒ provide S3 location for save query results



3. Run queries(same as hive)

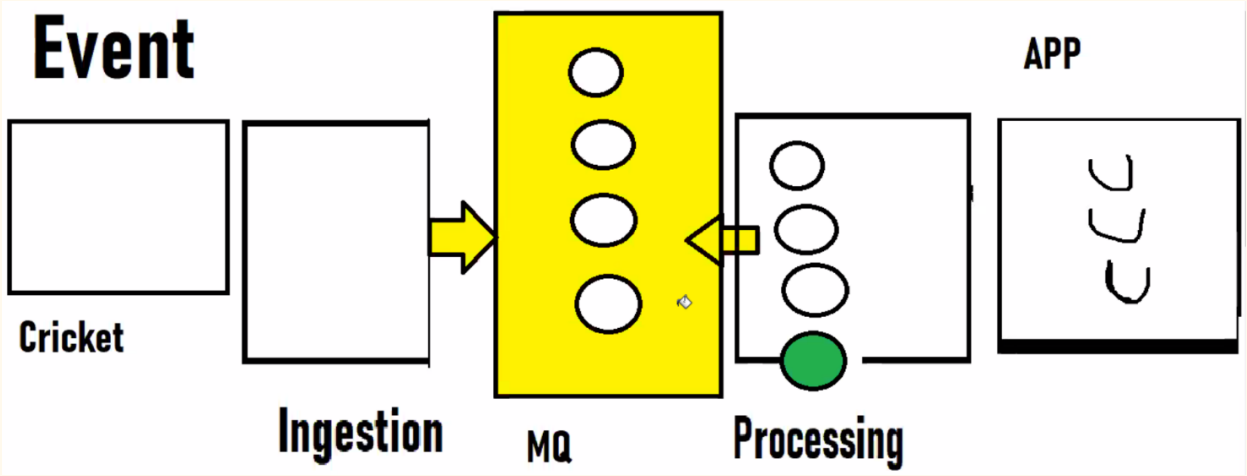


Kafka - 2012

Kafka is Message queue(MQ) tool between publisher and consumer.

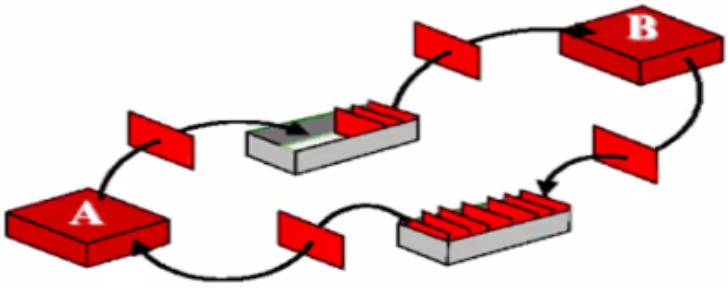
- Kafka is open source, distributed, MQ
- It follows Publisher/consumer model - you have to publish data to kafka and you have to find a way to consume data from Kafta.
- Kafka by default store data in MQ for **168 hours.** We can change this setting in conf file: **kafka/config/server.properties** then change the below setting -

```
log.retention.hours=168
```

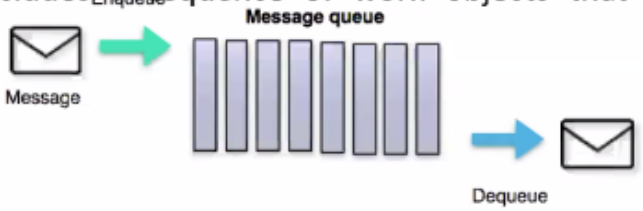


Introduction to Message Queue

Message queuing allows applications to communicate by sending messages to each other. The message queues provides a **temporary message storage** when the destination program is **busy or not connected**.



A **queue** is a line of things waiting to be handled - in sequential order starting at the beginning of the line. A message queue is a queue of messages sent between applications. It includes a sequence of work objects that are waiting to be processed.



Kafka Introduction

- Kafka is a leading general purpose ***publish-subscribe distributed messaging system***, which offers strong durability, scalability and fault-tolerance support.

- Designed for handling of ***real time activity*** stream data such as logs, metrics collections.

- Written in ***Scala***

- An apache project initially developed at ***LinkedIn*** at the year 2007.

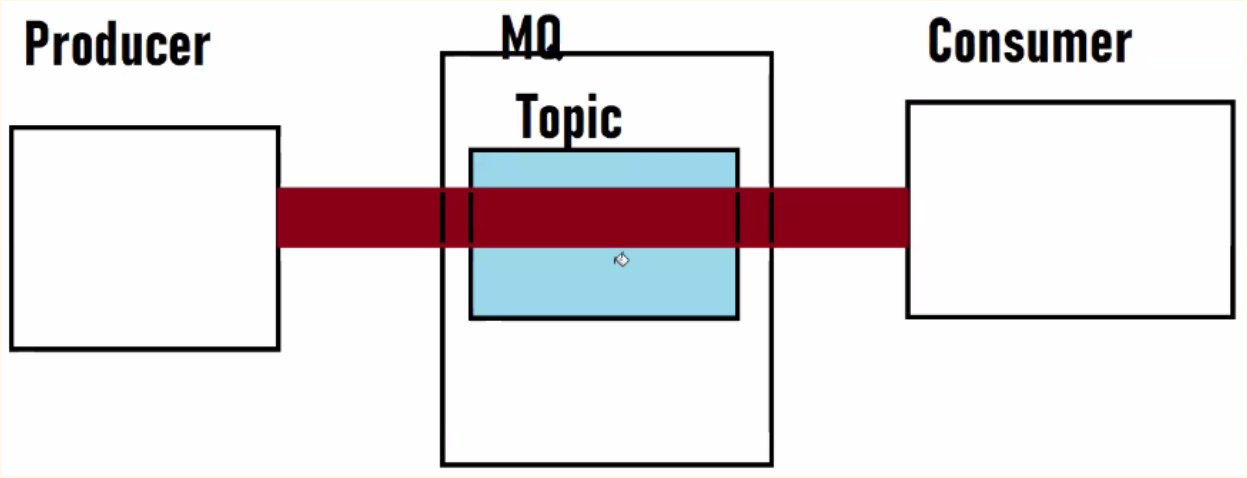
- It is **not specifically designed for Hadoop** rather Hadoop ecosystem is just be one of its possible consumers.

Why need Kafka

- Kafka is Highly scalable very *easy to add large number of consumers*
- Kafka handle spike
- Kafka also supports different consumption model
- Message durability is high in Kafka – *Persists*
- Integration support for Kafka with other frameworks are high

How to start Kafka

When kafka starts a new "**Topic**" created. **Topic** is like new tunnel insida Message Queue.



Install Kafka and How to start Kafka

1. Download zookeeper and Kafka - (make sure to download binary files instead of src files)
    - Zookeeper - https://archive.apache.org/dist/zookeeper/zookeeper-3.4.12/zookeeper-3.4.12.tar.gz
    - Kafka - https://archive.apache.org/dist/kafka/0.11.0.3/kafka_2.11-0.11.0.3.tgz

      For ubuntu/mac follow an article or in above set-up update .bat with .sh
    - Mac Kafka - https://medium.com/@Ankitthakur/apache-kafka-installation-on-mac-using-homebrew-a367cdefd273
    - Ubuntu - https://tecadmin.net/how-to-install-apache-kafka-on-ubuntu-22-04/
2. Start zookeeper ->
   Zookeeper ⇒ conf folder ⇒ rename zoo_sample.cfg to zoo.cfg
3. Zookeeper ⇒ conf folder ⇒ open terminal ⇒ type ./zkserver ⇒ then zookeeper will start
4. Go inside Kafka folder ⇒ open the terminal and type the below command and Kafka will start -

   ```
   .\bin\windows\kafka-server-start.bat .\config\server.properties
   ```
5. To create "**Topic":** kafka folder ⇒ bin ⇒ windows folder ⇒ open terminal and then run below command -

   ```
   kafka-topics.bat --create --zookeeper localhost:2181
   --replication-factor 1 --partitions 1 --topic topicName
   ```
6. To start **producer console** :--> kafka folder ⇒ bin ⇒ windows folder ⇒ open terminal and then run below command -

   ```
   kafka-console-producer.bat --broker-list localhost:9092 --topic
   topicName
   ```
7. To start **consumer console** :--> kafka folder ⇒ bin ⇒ windows folder ⇒ open terminal and then run below command

   ```
   kafka-console-consumer.bat --zookeeper localhost:2181  --topic
   topicName --from-beginning
   ```

8. Now produce some data in **producer console** and heck output in **consumer console.**

Kafka Spark Integration

Spark need 6 steps to connect with kafka -

1. Remove tmp folder then
   - ➔ Start Zookeeper
   - ➔ start kafka
   - ➔ Create a topic newtp
   - ➔ Open Eclipse/Intellij
   - ➔ Use existing project or Create new Project
2. Kafka spark connector jar
3. Necessary import statements
4. val ssc = new StreamingContext(conf, seconds(2))
5. val topicName = Array("newTopic")
6. val kafkaParams = Kafka configuration params

   Here kafka server is localhost:9092

```
val kafkaParams = Map[String, Object]("bootstrap.servers" -> "localhost:9092",
    "key.deserializer" -> classOf[StringDeserializer],
    "value.deserializer" -> classOf[StringDeserializer],
    "group.id" -> "example",
    "auto.offset.reset" -> "latest"
    )
```

7. val stream = KafkaUtils(ssc, topicName, kafkaParams)

```
val stream = KafkaUtils.createDirectStream(
 ssc, PreferConsistent, Subscribe[String, String](
   topics, kafkaParams))
 .map(x => x.value())
 .map(x => x.concat("===,sai")) /*Transformation if want to do over streaming
data*/
```

Complete code for kafka integration

```
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.streaming.StreamingContext
```

```scala
import org.apache.spark.streaming._
import org.apache.spark.sql._
import org.apache.spark.sql.functions
import org.apache.kafka.clients.consumer._
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

object kafka_Integration_spark {

 def main(arg: Array[String]): Unit = {

   val conf = new SparkConf().setAppName("Spark Project").setMaster("local[*]")
     .set("spark.driver.allowMultipleContexts", "true") /* As streamingContext is also uses sparkContext,
so using this it can also run multiple sparkContext*/

   val sc = new SparkContext(conf)
   val spark = SparkSession.builder().getOrCreate()
   sc.setLogLevel("error")

   val ssc = new StreamingContext(conf, Seconds(2))  /* every 2 sec spark will consume data from kafka */


   val topics = Array("topic1")

   val kafkaParams = Map[String, Object](
     "bootstrap.servers" -> "localhost:9092",   /* Local server */
     "key.deserializer" -> classOf[StringDeserializer],
     "value.deserializer" -> classOf[StringDeserializer],
     "group.id"    ->     "example",  /* Consumer Name */
     "auto.offset.reset" -> "latest"    /* consumption model type */
   )

   val stream = KafkaUtils.createDirectStream[String, String](
     ssc, PreferConsistent, Subscribe[String, String](topics,kafkaParams))
    .map(x => x.value())
    .map(x => x.concat("===,sai")) /* Transformation if want to do over streaming data */

   stream.print
   ssc.start()  /* Need to start streaming - mandatory */
   ssc.awaitTermination()
 }
}
```

Task

- Push the data to Kafka CLI
- Consume from Spark
- Convert to dataframe
- Add new column and attach current timestamp
- Write data to RDBMS

```
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming._
import org.apache.spark.sql._
import org.apache.spark.sql.functions
import org.apache.kafka.clients.consumer._
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.sql.functions._

object obj {
   def main(args:Array[String]):Unit={
      val conf = new SparkConf().setAppName("first").setMaster("local[*]")
           .set("spark.driver.allowMultipleContexts","true")
      val sc = new SparkContext(conf)
      sc.setLogLevel("ERROR")
      val spark= SparkSession.builder.getOrCreate()
     import spark.implicits._
     val ssc = new StreamingContext(conf,Seconds(2))
     val topics = Array("ztk")
     val kafkaParams = Map[String, Object](
      "bootstrap.servers" -> "localhost:9092",
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[StringDeserializer],
      "group.id" -> "gp2",
      "auto.offset.reset" -> "latest")

      val stream = KafkaUtils.createDirectStream[String, String](
                  ssc,PreferConsistent,Subscribe[String, String](
```

```
                topics, kafkaParams)).map( x => x.value())

    stream.foreachRDD(x=>
      if(!x.isEmpty()){
            val df = x.toDF("name").withColumn("time1",current_timestamp)
            df.show()
           } )
      ssc.start()
      ssc.awaitTermination()
      }
}
```

Kafka Consumption Model

1. **Latest** - Consumers will only take latest coming data from kafka. Suppose one topic have produces lots of data but we want only **latest** data, Then How to get only latest data in different tools -
   - ➔ **Spark** ⇒ we can simply give new group.id and consumption model in spark by adding param `"auto.offset.reset" -> "latest"`

     ```
     val kafkaParams = Map[String, Object](
           "bootstrap.servers" -> "localhost:9092",
          "key.deserializer" -> classOf[StringDeserializer],
          "value.deserializer" -> classOf[StringDeserializer],
          "group.id" -> "gp1",
          "auto.offset.reset" -> "latest"
     )
     ```

   - ➔ **Kafka CLI(consumer CLI)** ⇒ only add this `--from-beginning` at the end of command when starting consumer.

     ```
     bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
     --topic testTopic --from-beginning
     ```

2. **Earliest** - Consumer will all oldest from kafka

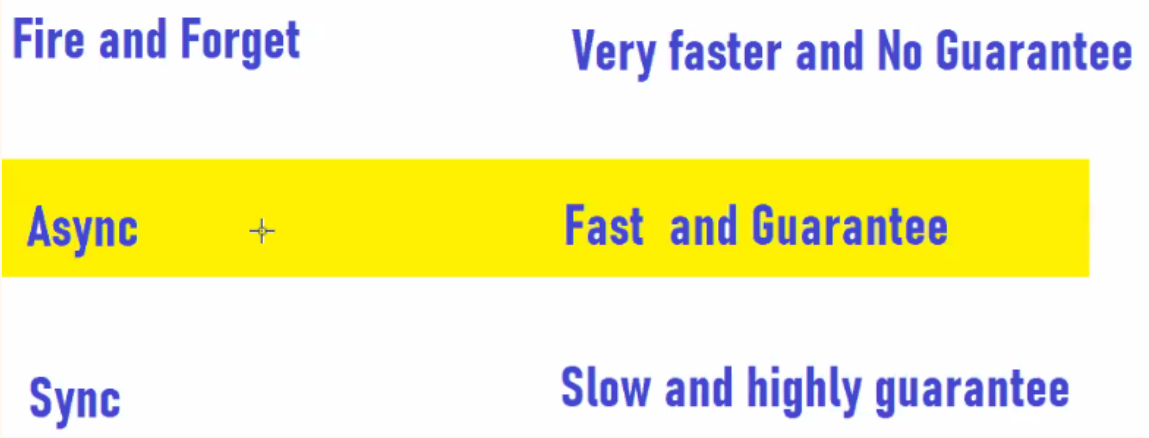
   ```
   "group.id" -> "gp2",
   "auto.offset.reset" -> "earliest"
   ```

Kafka UI

Download Offset Explorer - https://www.kafkatool.com/download.html

ISR (Insync Replica)

1. Fire and Forget - producer have send the message but didn't wait for acknowledgment
2. Async - producer have sent the message and received the acknowledgment
3. Sync - producer have sent the message and received the acknowledgment from all replica nodes.



Cloud MQ

1. Kinesis - AWS
2. Event Hub - Azure
3. Google MQ

Kinesis Consumption Model

1. **Latest** - Consumer will only take latest coming data from kafka. Suppose one topic have produces lots of data but we want only **latest** data, Then How to get only latest data in different tools -
   ➔ **Spark** ⇒ consumption model in spark by adding param `InitialPositionInStream.LATEST`

```
val stream= KinesisUtils.createStream(ssc,
                        "zeyogroup",
                        "MQ_NAME",    // MQ_NAME
                        "https://kinesis.ap-south-1.amazonaws.com",
                        "ap-south-1",
                        InitialPositionInStream.LATEST,
                        Seconds(2),
                        StorageLevel.MEMORY_AND_DISK_2)
```

2. **Trim_Horizon** - Consumer will all oldest from kafka
   `InitialPositionInStream.Trim_Hprizon`

3. **AT_TIMESTAMP** ⇒ Suppose there are so many data in MQ but you want data only fro specific timestamp, then use this model
   `InitialPositionInStream.AT_TIMESTAMP`

Kinesis Integration with Spark

1. **Create kinesis MQ in AWS or run below command from CLI if AWS is configured in CLI**

```
aws kinesis create-stream  --stream-name ayushi --shard-count 1
```

2. **Create project ⇒ Add kinesis jars**
3. **Initialize kinesis streaming object**
4. **Build `KinesisInputDStream` or create `KinesisUtils` object**

```
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.kinesis.{KinesisInputDStream, KinesisUtils}
import org.apache.spark.streaming.{Seconds, StreamingContext}
```

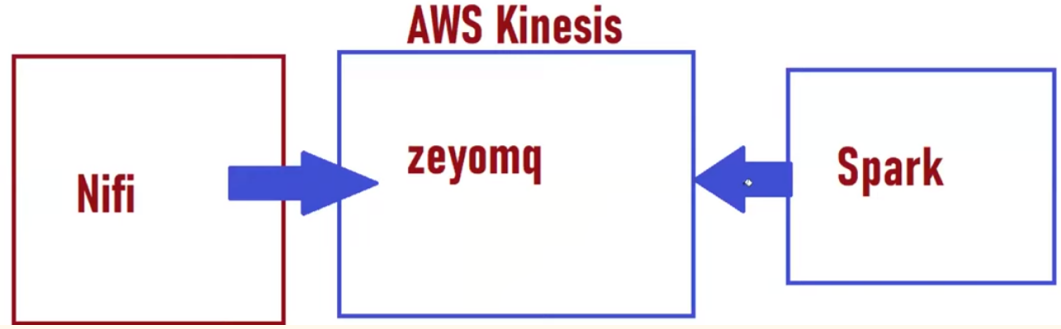
```
object obj {
  def byteToString(a: Array[Byte]): String = new String(a)

  def main(args:Array[String]):Unit={
    val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                  .set("spark.driver.allowMultipleContexts","true")
                  .set("AWS_ACCESS_KEY","AKIAV6SZ74UI62JECW6X")
                  .set("AWS_SECRET_KEY","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")
                  .set("AWS_CBOR_DISABLE","true")

      val sc = new SparkContext(conf)
      sc.setLogLevel("ERROR")
      val spark = SparkSession.builder().getOrCreate()
      import spark.implicits._

      val ssc = new StreamingContext(conf,Seconds(2))

    // 1st way to create kinesis dstream object
    val kinesisStream = KinesisInputDStream.builder.streamingContext(ssc)
                        .endpointUrl("https://kinesis.ap-south-1.amazonaws.com")
                        .regionName("ap-south-1")
                        .streamName("MQ_Name") // MQ_NAME
                        .initialPositionInStream(InitialPositionInStream.LATEST)
                        .checkpointAppName("zeyo1234group")
                        .checkpointInterval(Seconds(2)) //Spark will consume data from kinesis in
every 2 sec
                        .storageLevel(StorageLevel.MEMORY_AND_DISK_2)
                         .build()

      // 2nd way to create kinesis dstream object
      val stream= KinesisUtils.createStream(ssc,
                        "zeyogroup",
                        "MQ_NAME",    // MQ_NAME
                        "https://kinesis.ap-south-1.amazonaws.com",
                        "ap-south-1",
                        InitialPositionInStream.LATEST,
                        Seconds(2),
                        StorageLevel.MEMORY_AND_DISK_2)

      val finalstream=stream.map(x => byteToString(x)) //convert byte stream data to string
      finalstream.print()
      ssc.start()
      ssc.awaitTermination()
```

```
    }
}
```

5. **Run spark Application**
6. **Put data from publisher console using below command and check same data is comming in consumer console(Spark application)**

```
aws kinesis put-record --stream-name YOUR_MQ_NAME --partition-key 123
--data zeyobron
```

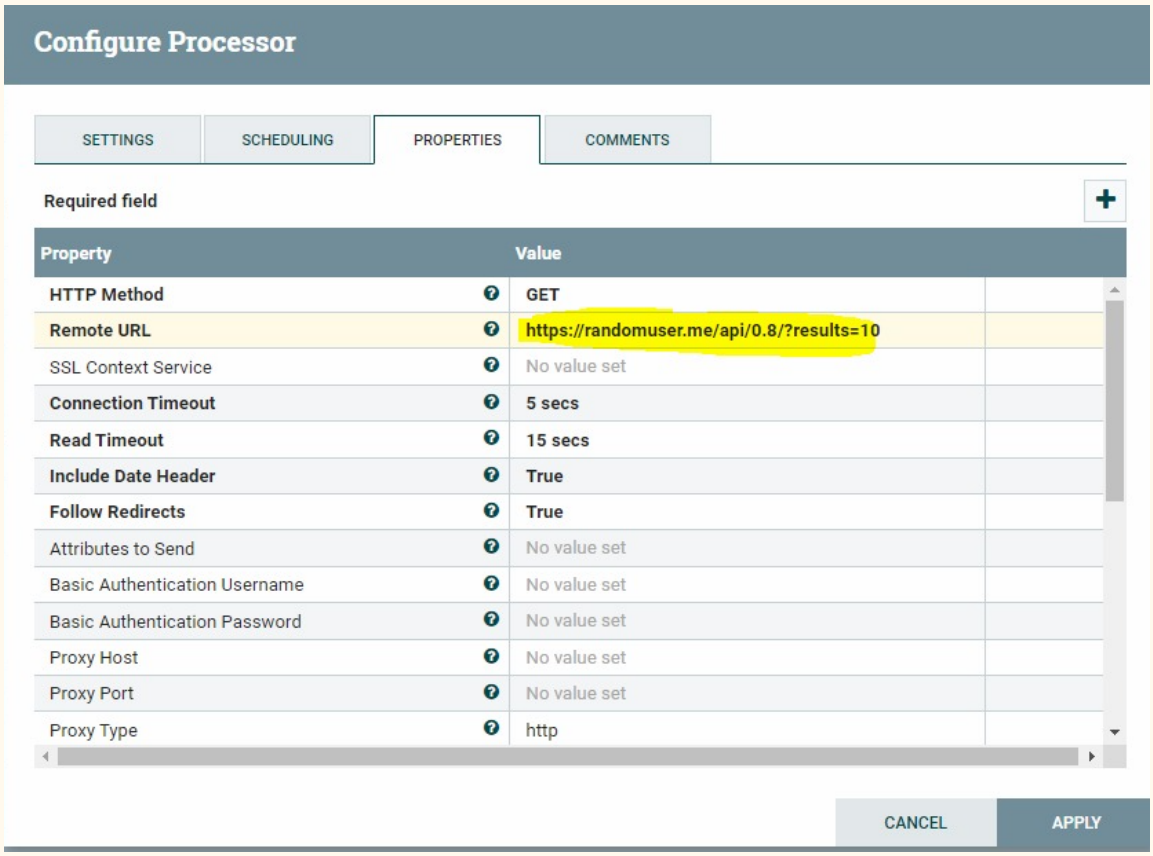
Steps to set up kinesis flow from Nifi to Spark



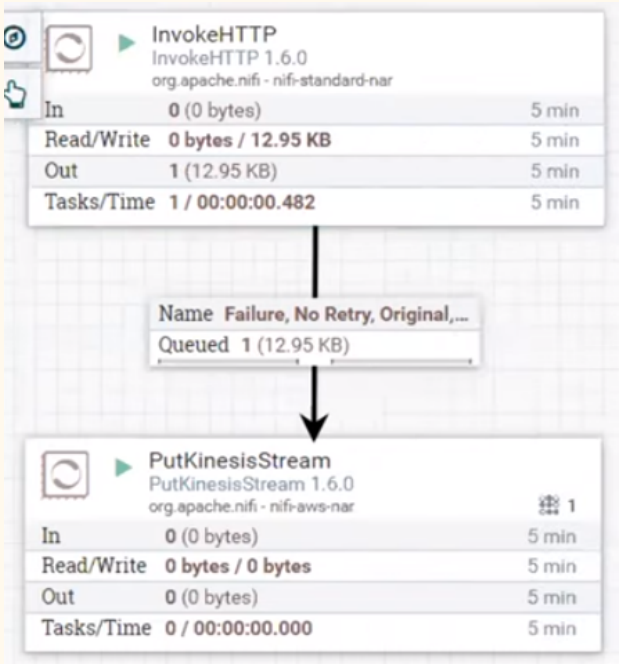
7. **Create kinesis MQ in AWS or run below command from CLI if AWS is configured in CLI**

```
aws kinesis create-stream  --stream-name ayushi --shard-count 1
```

8. **Consumer ⇒ Go to Nifi and create flow, first invoke https data from URL -**



9. **Configure Kinesis in Nifi =>** create put kinesis module and set below config , after that connect both module **invokeHttp** and **putKinesis**

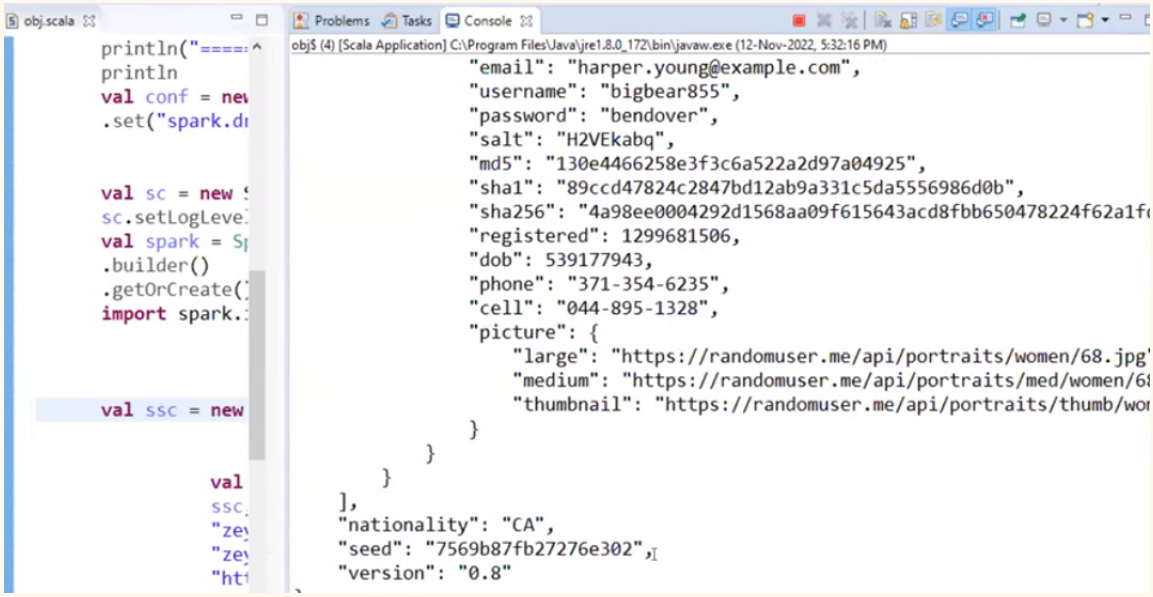


10. As consumer configure kinesis with Spark

```
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.kinesis.{KinesisInputDStream, KinesisUtils}
import org.apache.spark.streaming.{Seconds, StreamingContext}


object obj {
  def byteToString(a: Array[Byte]): String = new String(a)

  def main(args:Array[String]):Unit={
    val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                .set("spark.driver.allowMultipleContexts","true")
                .set("AWS_ACCESS_KEY","AKIAV6SZ74UI62JECW6X")
                .set("AWS_SECRET_KEY","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")
                .set("AWS_CBOR_DISABLE","true")

      val sc = new SparkContext(conf)
      sc.setLogLevel("ERROR")
      val spark = SparkSession.builder().getOrCreate()
      import spark.implicits._
```

```
      val ssc = new StreamingContext(conf,Seconds(2))

    // 1st way to create kinesis dstream object
    val kinesisStream = KinesisInputDStream.builder.streamingContext(ssc)
                        .endpointUrl("https://kinesis.ap-south-1.amazonaws.com")
                        .regionName("ap-south-1")
                        .streamName("MQ_Name") // MQ_NAME
                        .initialPositionInStream(InitialPositionInStream.LATEST)
                        .checkpointAppName("zeyo1234group")
                        .checkpointInterval(Seconds(2))
                        .storageLevel(StorageLevel.MEMORY_AND_DISK_2)
                         .build()

      // 2nd way to create kinesis dstream object
      val stream= KinesisUtils.createStream(ssc,
                        "zeyogroup",
                        "MQ_NAME",   // MQ_NAME
                        "https://kinesis.ap-south-1.amazonaws.com",
                        "ap-south-1",
                        InitialPositionInStream.LATEST,
                        Seconds(2),
                        StorageLevel.MEMORY_AND_DISK_2)

      val finalstream=stream.map(x => byteToString(x)) //convert byte stream data to string
      finalstream.print()
      ssc.start()
      ssc.awaitTermination()
      }
}
```

11. Run spark application and you will get result in console



NoSQL

- Open source
- Dynamic schema
- Horizontal scaling
- Object Querying - if data is in JSON and need only 1 attribute, we can easily query that data in NoSQL

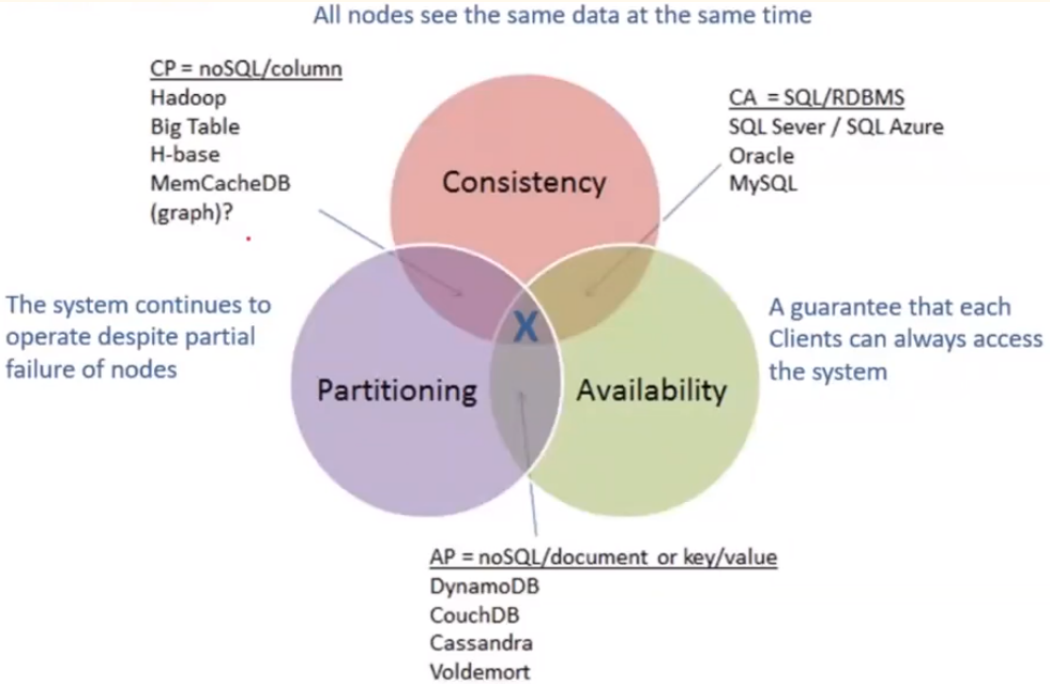
NoSQL DBs

- Big table - Google launched in 2005, it is fist NoSQL DB
- HBase, Cassandra, MongoDB, couchDB, couch, DynamoDB, cosmo, vertica, openSearch, ElasticSearch

Ques - Which NoSQL should use?

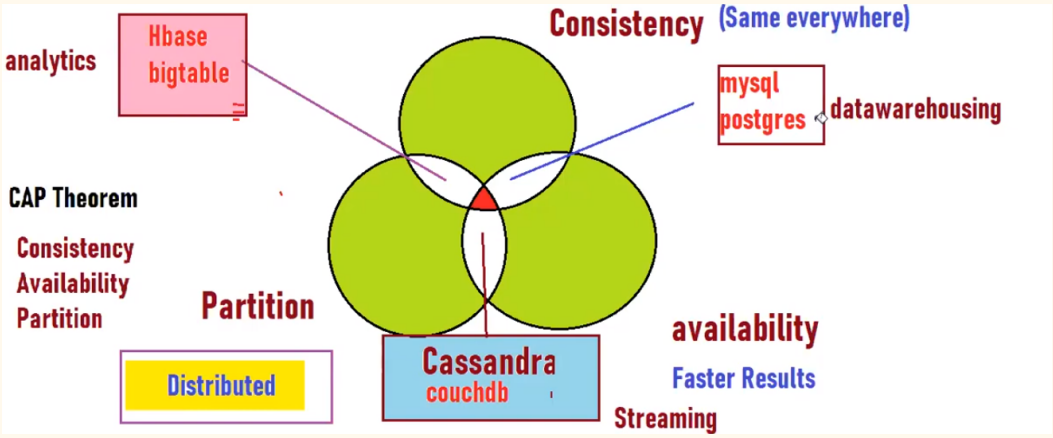
Ans - Follow Cap Therom and select NoSQL DB accordingly

Cap Therom



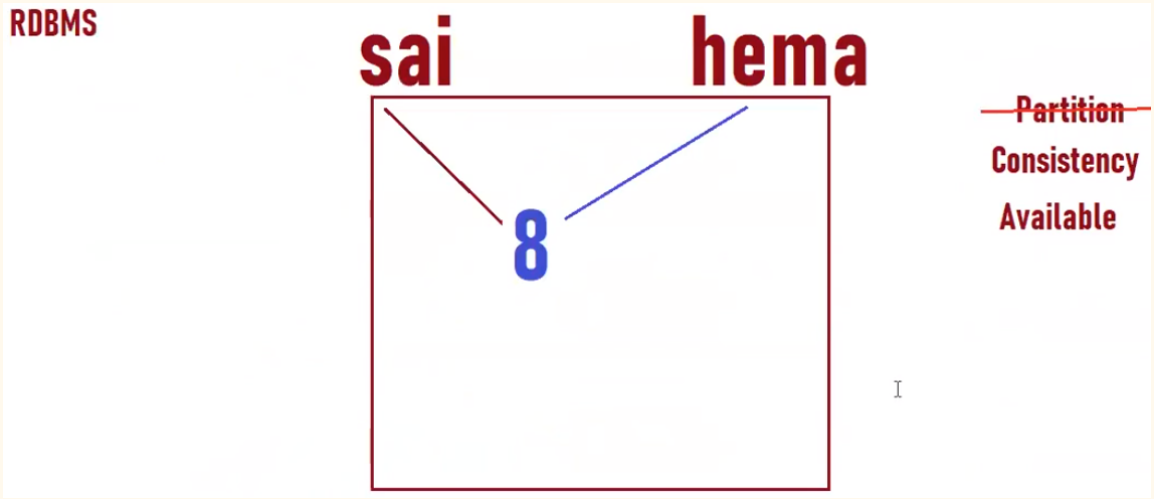
1. **Consistency**. Consistency means that the user should be able to see the same data no matter which node they connect to on the system. This data is the most recent data written to the system. So if a write operation has occurred on a node, it should be replicated to all its replicas. So that whenever a user connects to the system, they can see that same information.
2. **Availability**. All reads contain data, but it might not be the most recent. Whether the user wants to read or write, the user should get a response even if the operation was unsuccessful.
3. **Partition tolerance.** Partition refers to a communication break between nodes within a distributed system. Meaning, if a node cannot receive any messages from another node in the system, there is a partition between the two nodes. Partition could have been because of network failure, server crash, or any other reason.

   So, if Partition means a break in communication then Partition tolerance would mean that the system should still be able to work even if there is a partition in the system. Meaning if a node fails to communicate, then one of the replicas of the node should be able to retrieve the data required by the user.



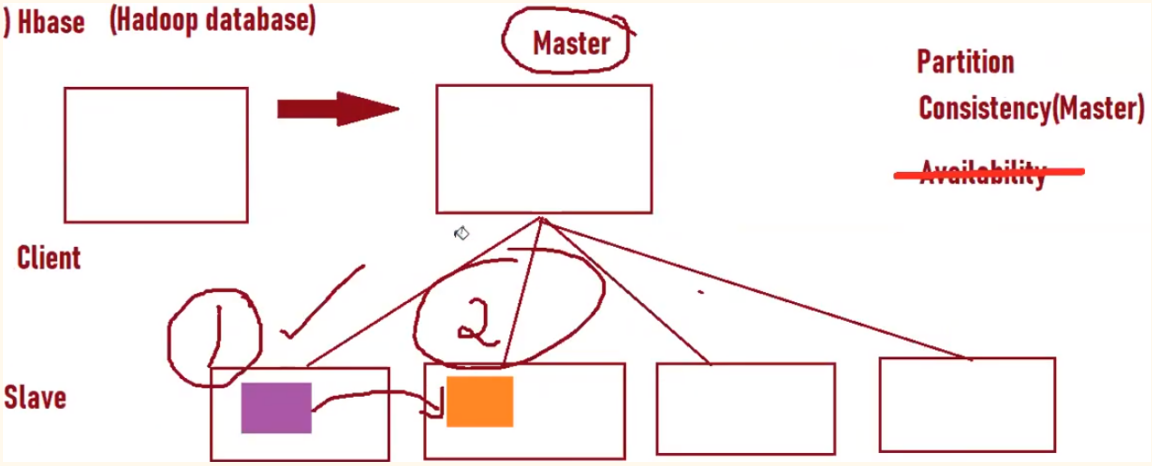
RDBMS

Single server architechure



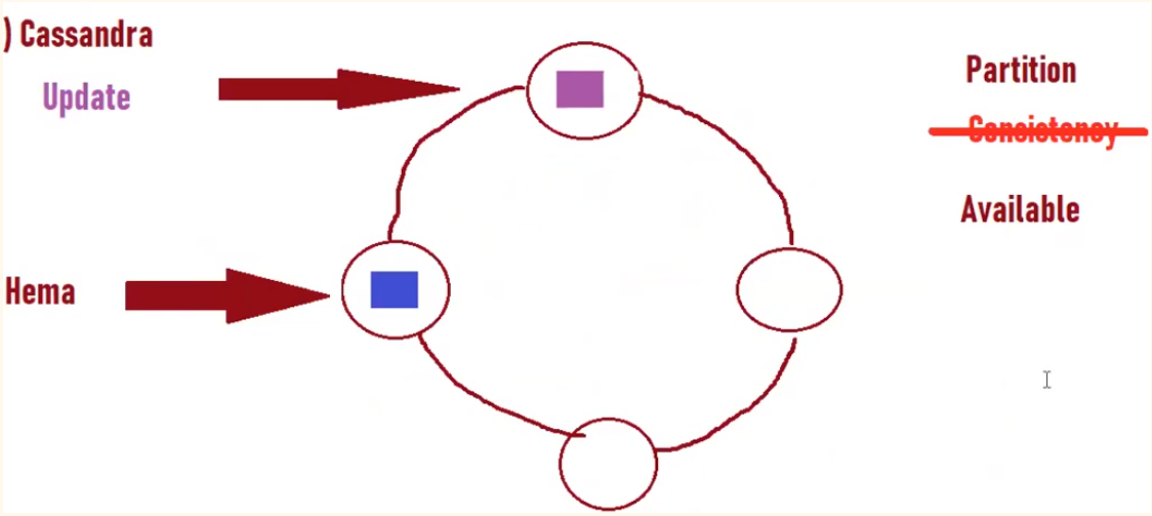
HBase

Follow maste slave architecture



Cassandra

Every node is master and every node is salve, it follow ring acrchitecture



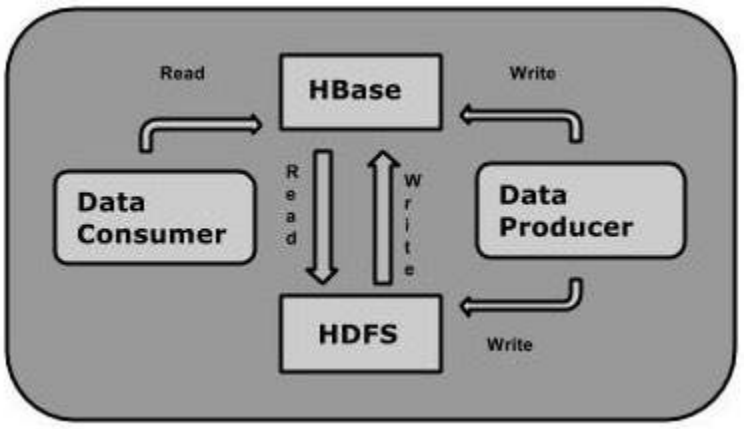
Cassandra commands

```
systemctl status cassandra
====== cqlsh =================
cqlsh --request-timeout=3600000

---> Specify a remote node IP address:
cqlsh 10.0.0.30
```

```
--->> to save the output of a SELECT statement to myoutput.txt:
cqlsh -e "SELECT * FROM mytable" > myoutput.txt

-----> copy data from table -
 copy tableName(id,name,job,phone_no) To 'myfile.txt'

---- > copy data from file into table
copy personal(id) from 'output.txt' with header=true;

---> copy all data in to one file from different tables;
capture 'output.txt'
Select * from keyspaceNmae.tableName;
capture off;

---> Expand :This command is used to expand the output.
expand on;
select * from tableName;
expand off;
================================================================================
SELECT * FROM system_schema.keyspaces;
Create keyspace KeyspaceName with replication={'class':'strategy name',
'replication_factor':1};
use keyspaceName;
CREATE TABLE tableName(columnName DATATYPE....);
INSERT INTO tableName(ColumnName,ColumnName,..) VALUES(value,value,...);
describe tables;

ALTER TABLE [keyspace_name.]table_name
[ADD (column_definition_list)]
[DROP column_list]
[RENAME column_name TO column_name];

ALTER KEYSPACE keyspace1 WITH REPLICATION =
  {'class' : 'NetworkTopologyStrategy', 'dc1' : 0, 'dc2' : 3, 'dc3' : 0 };

CREATE INDEX indexName ON tableName(ColumnName);
DROP INDEX indexName;

TRUNCATE person.details ;
DELETE FROM tableName where id=101;
DROP TABLE tableName;
DROP KEYSPACE kepspaceName;
```

HBase

- HBase is Open Source, distributed Publish subscribe Message queue.
- it runs on top of HDFS (Hadoop distributed file system).
- Habse data stores in HDFS, we can access Hbase using ⇒ **hadoop fs -ls /hbase**
- HBase doesn't require column name while creating table
- HBase provides capabilities like Bigtable which contains billions of rows and millions of columns to store the vast amounts of data. It allows users to save billions of records and retrieves the information instantly. For example, the HBase consists of 5 billion records, and in that, if you wish to find 20 large items, HBase does it for you immediately: that's how it works.



Limitations of Hadoop and the Emergence of HBase

- Hadoop is capable of only batch processing, and it requires access to data in a sequential manner, which means, one has to search entire data to get the specific information.
- In Hadoop, it is required to process enormous data sets and the result is, we get is the same huge data sets that again need to be processed. This caused a redundancy and therefore needed a solution to access the specific point of data.

Where to Use HBase?

- Hadoop HBase is used to have random real-time access to the Big data.
- It can host large tables on top of cluster commodity.
- HBase is a non-relational database which modeled after Google's big table. It works similarly to a big table to store the files of Hadoop.

Limitations of HBase

- It takes a very long time to recover if the HMaster goes down. It takes a long time to activate another node if the first nodes go down.
- In HBase, cross-data operations and join operations are very difficult to perform, even if we join operations by using MapReduce, it requires a lot of time to design and develop.
- HBase is expensive in terms of hardware requirement and memory blocks' allocations.

Ques - Where does HBase data stores in Hadoop?

HBase stores on HDFS we can find HBase directories under

```
hadoop fs -ls /hbase
```

HBase Terminology

1. Table name
2. Column family
3. Row key
4. Column name
5. value

   Notes -

   ➔ **Hbase doesn't have primary key, But here row key is unique and will work as a primary key**

   ➔ **Same column name can exist in different column families in the same table**

```
put 'htab','1','columnFamily1:pname','zeyo'
'columnFamily2:pname','xyz'
```

Start HBase

1. Login to cloudera and type ⇒

```
sudo service hbase-master restart
hbase shell
```

2. Run hbase commands -

```
list // show all tables
create 'tableName','columnFamilyName'  // create table
put 'htab','rowKey','columnFamilyName:columnName','value'  // insert data
put 'htab','1','zcf:pid','p1'  // insert data
scan 'htab'  // describe table
```

3. Once we do scan on the table, table comes like below ⇒

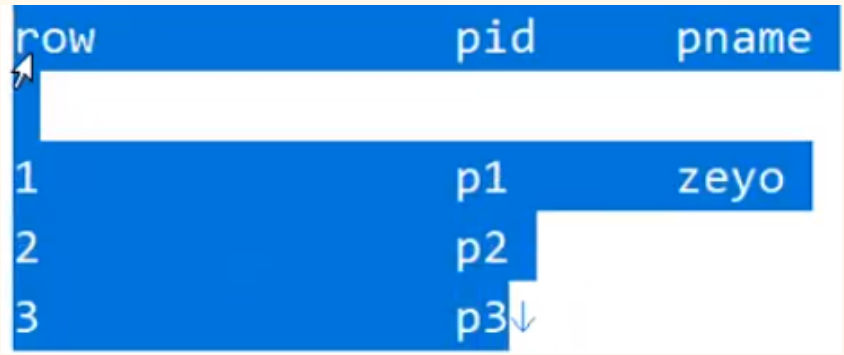
```
hbase(main):008:0> scan 'htab'
ROW                   COLUMN+CELL
 1                    column=zcf:pid, timestamp=1668343445236, value=p1
 2                    column=zcf:pid, timestamp=1668343618468, value=p2
2 row(s) in 0.0490 seconds
```

4. **Suppose we do another insert with a new column for existing data, here pname is new column and we are doing insert for rowKey=1 which is existing data, it means if we do count of row there should be 3 rows only -**

```
put 'htab','1','zcf:pname','zeyo'
```

```
hbase(main):012:0> scan 'htab'
ROW                   COLUMN+CELL
 1                    column=zcf:pid, timestamp=1668343445236, value=p1
 1                    column=zcf:pname, timestamp=1668343730480, value=zeyo
 2                    column=zcf:pid, timestamp=1668343618468, value=p2
 3                    column=zcf:pid, timestamp=1668343665391, value=p3
3 row(s) in 0.0230 seconds
```

A new column will be created like below representation ⇒



How to create Hive table on top of Hbase table

```
create external table hbasehive(hkey string, hid string, hname string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties ("hbase.columns.mapping" =
":key,columnFamilyName:columnName,columnFamilyName:columnName")
tblproperties("hbase.table.name"="hbaseTableName");
```

For example ⇒

```
create external table hbasehive
(hkey string, hid string,hname string,hdata double) STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with serdeproperties
("hbase.columns.mapping" = ":key,zcf:pid,zcf:pname,zcf:pdata)
tblproperties("hbase.table.name"="htab");
```

Then do **select * from hbasehive;**

```
    > select * from hbasehive;
OK
1       p1      zeyo    NULL
2       p2      NULL    NULL
3       p3      NULL    NULL
4       NULL    NULL    sai
Time taken: 3.85 seconds, Fetched: 4 row(s)
hive>
```

Notes - If we do any insert in hive table, then data will also be reflected in Habse as well as both tables are representing same table.

Same goes with Hbse, if hbase table data is increased then that data will also reflect in Hive.

HBase integration with HBase

1. Run ⇒ spark-shell

   These steps only for adding jars in cloudera

   - create a folder in edge Node (cloudera or Lab)
   - Copy those HBase jars to the folder
   - Then run command ⇒
     `spark-shell --conf "spark.driver.extraClassPath=/home/cloudera/hbasejars/*"`

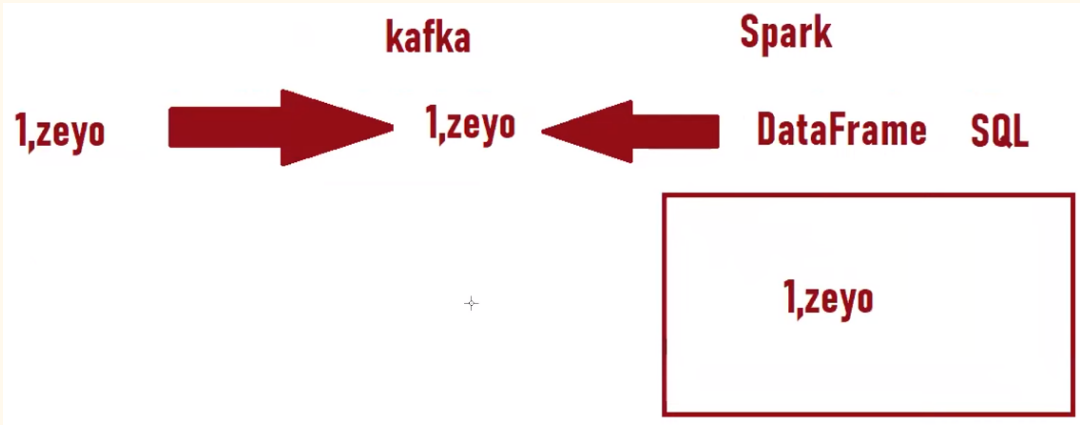
2. Add Hbase jar using ⇒ `spark-shell --package <artifact-id> <>`
3. Then sun spark code in shell ⇒

```
import org.apache.spark.sql.{SQLContext, _}
import org.apache.spark.sql.execution.datasources.hbase._
import org.apache.spark.{SparkConf, SparkContext}
import spark.implicits._
def catalog = s"""{
    |"table":{"namespace":"default", "name":"htab"},
    |"rowkey":"masterid",
    |"columns":{
    |"srow":{"cf":"rowkey", "col":"masterid", "type":"string"},
    |"sid":{"cf":"zcf", "col":"pid", "type":"string"},
    |"sname":{"cf":"zcf", "col":"pname", "type":"string"},
    |"sdata":{"cf":"zcf", "col":"pdata", "type":"string"}
    |}
|}""".stripMargin

val df = spark.read.format("org.apache.spark.sql.execution.datasources.hbase")
          .options(Map(HBaseTableCatalog.tableCatalog->catalog)).load()

df.show()
```

```
scala> df.show( )
+----+----+-----+-----+
|srow| sid|sname|sdata|
+----+----+-----+-----+
|   1|  p1| zeyo| null|
|   2|  p2| null| null|
|   3|  p3| null| null|
|   4|null| null|  sai|
+----+----+-----+-----+
```

Spark Dataframe Streaming

- In spark **2.0**, we can directly put stream data in **Dataframe streaming(Structured streaming).**
- Before it, we have put to stream data in **RDD(DStream)** => convert in Spark Dataframe.

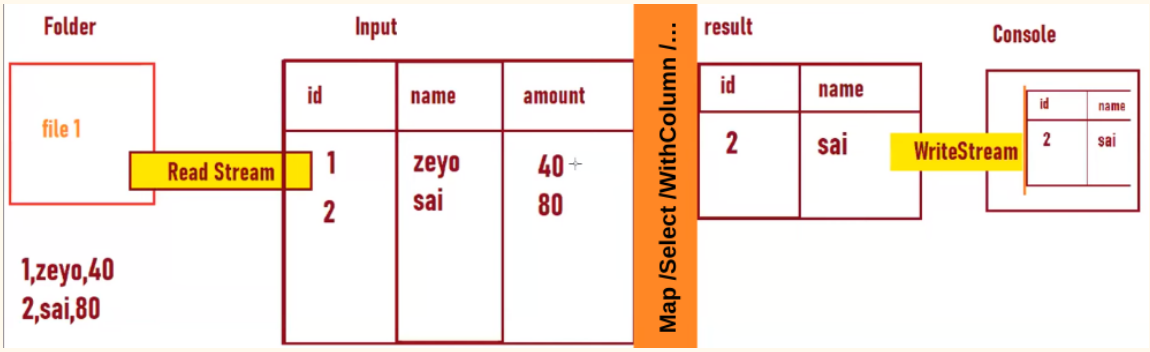


Stream Project example

1. Create structured stream for a file path
2. Put data file in that folder

```
1,zeyo
2,sai
```

3. Took data from that path in every 2 sec using **readStream**
4. Then show data in the console using **writeStream**
5. Repeat steps 2-4 for better results




```
package pack
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._

object obj {
case class zeyoschema(id:String,category:String,product:String,mode:String)

def main(args:Array[String]):Unit={
     val conf = new SparkConf().setAppName("first").setMaster("local[*]")
     val sc = new SparkContext(conf)
     sc.setLogLevel("ERROR")
      val spark = SparkSession.builder().getOrCreate()
      import spark.implicits._
      val schema1 = StructType(Array(
                        StructField("id",StringType,true),
                        StructField("name",StringType,true)
                        ))

      val df = spark.readStream.format("csv")
                  .schema(schema1).load("file:///C:/input/data")
                  .filter("id==2").select("id", "name")

      df.writeStream.format("console")
              .option("checkpointLocation", "file:///C:/data/c1h12k")
//checkpointLocation have metadata of streaming like offsets, commits, like
when data come, from which source data some
```

```
            .start()
            .awaitTermination()
   }
}
```

Kafka integration with Structured streaming

- Start zookeeper
- Start Kafka
- Create or Use existing Topic of Kafka
- Add Kafka jars to the Project
- Put the Code (Change the Topic accordingly)
- Run the code
- Push the data
- Verify in Spark

```
package pack
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {

def main(args:Array[String]):Unit={
      val conf = new SparkConf().setAppName("first").setMaster("local[*]")
      val sc = new SparkContext(conf)
      sc.setLogLevel("ERROR")

      val spark = SparkSession.builder().getOrCreate()
      import spark.implicits._

      val kafkadf = spark.readStream.format("kafka")
                  .option("kafka.bootstrap.servers", "localhost:9092")
                  .option("subscribe","stk")
                  .load()
                  .withColumn("value", expr("CAST(value as string)"))
                  .select("value")

      kafkadf.writeStream.format("console")
```

```
            .option("checkpointLocation", "file:///C:/data/kdata")
            .start()
            .awaitTermination()
    }
}
```

Project #2 - take streaming data from Kafta(topic1) and write that data back to kafka(topic2)

```
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {

def main(args:Array[String]):Unit={
      val conf = new SparkConf().setAppName("first").setMaster("local[*]")
      val sc = new SparkContext(conf)
      sc.setLogLevel("ERROR")
      val spark = SparkSession.builder().getOrCreate()
      import spark.implicits._

    // read stream data from kafka topic1
      val kafkadf = spark.readStream.format("kafka")
                  .option("kafka.bootstrap.servers", "localhost:9092")
                  .option("subscribe","topic1").load()
                  .withColumn("value", expr("CAST(value as string)"))
                  .select("value")
                  .withColumn("value", expr("concat(value,'~','sai')"))

      // write stream data back to kafka topic2
      kafkadf.writeStream.format("kafka")
          .option("kafka.bootstrap.servers", "localhost:9092")
          .option("topic", "topic2")
          .option("checkpointLocation", "file:///C:/data/kd1ata")
          .start().awaitTermination()
    }
}
```

Phase 3 - Project

Requirement ⇒

1. read streaming data from kafka
2. Write data to destination location(cassandra)

```
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.DataFrame

object obj {

def main(args:Array[String]):Unit={
      val conf = new SparkConf().setAppName("first").setMaster("local[*]")
      val sc = new SparkContext(conf)
      sc.setLogLevel("ERROR")
      val spark = SparkSession.builder().getOrCreate()
      import spark.implicits._

    // read stream data from kafka topic1
      val kafkadf = spark.readStream.format("kafka")
                  .option("kafka.bootstrap.servers", "localhost:9092")
                  .option("subscribe","topic1").load()
                  .withColumn("value", expr("CAST(value as string)"))
                  .select("value")
                  .withColumn("value", expr("concat(value,'~','sai')"))

    // write stream batch data to cassandra
      kafkadf.writeStream.foreachBatch{(df: DataFrame, id: Long) =>
            df.write.format("org.apache.spark.sql.cassandra")
                  .option("spark.cassandra.connection.host","localhost")
                  .option("spark.cassandra.connection.port","9042")
                  .option("keyspace","zeyodb")
                  .option("table","streamtable")
```

```
                    .save()
            }
            .option("checkpointLocation", "file:///C:/data/kd1ata")
            .start()
            .awaitTermination()
  }
}
```