Data Engineering

Mohan Kumar Pulibanti

Big Data Prerequisite Session:

Agenda – Day 1

1)Small Game

Fruits Game:
US and UK Selling Fruits to India, 4 Scenarios
As Data Increases, our Brains Can't suffice the size of data.
Brain – two parts – 1) HIPPOCampus(Stores) 2) PreFocalContext(Processes)
Storing – processing – Decision is All that happens in our day-to-day life

2) Big Data Introduction

A Friend – Wakes you up in the Morning, Helps to chat with friends, Helps to book cabs, Helps to become rich, financial Transactions…..So on

Friend is Data.

A Single User(with a Mobile, SmartWatch etc) is generating lot of Data. Where is this getting stored, How is this processed, How Decisions are taken based on this?

In the last 3 decades, lots of data has been generated. 7 billion smart phones of smart phones, each user generating 60 Exabytes of data per month.

Bit, Byte, KiloByte, MegaByte, GigaByte, TeraByte, petabyte, ExaByte, ZetaByte, Yottabyte, brontoByte, GeoByte

Google Search: 2.5 billion searches a day experience 2.5Exabytes of Data.
Netflix: 203 million hours of video streaming per day
YouTube: 4 million uploads per day
400 million emails per minute
Swiggy has 1.4 million orders a day
Zomato 1.5 million orders a day

How are these organizations handling this huge data, how are they processing it?

3)Analytics

The organization uses data to do analytics and grow business.
YouTube video Suggestions, Facebook showing Amazon products that we searched for or looking for.

4)Real Time Examples

Donald Trump won in 2016 with the help of Cambridge Analytica and Facebook. Facebook sold data related to Trump to Cambridge Analytica and Cambridge Analytica has done analytics and directed Trump to concentrate on that in his campaign.

Fruit sellers write the data on a paper to analyze what is to be bought tmrw.

Slides – Every Domain Data information (like Retailers – Amazon, Walmart; Finance - Visa)

5) Evolution of Data Analytics (Storage, Process, Decisions)

Data Analytics is all about Storage, Processing, and Making Decisions.

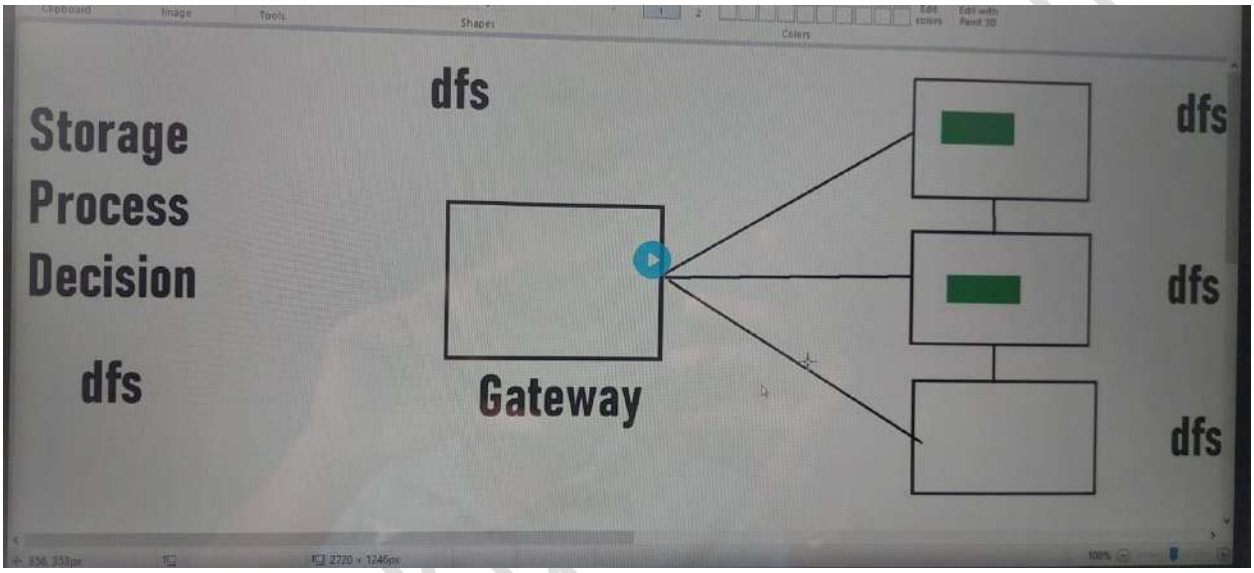
1960 – Computers are new, Txt files(Notepad) are used for Analytics
1970 – DBMS (Data Base Management System), better than Notepad.
Eg: Spreadsheet (they have rows and columns). DAN BRICKLIN launched a Spreadsheet for better calculations.
1980 – internet launched. Larry Ellison came forward and launched RDBMS (Relational database management system)- Oracle SQL.

he stated that users can store their data in RDBMS(oracle sql, mysql, postgres, netezaa,mariadb, aurora are examples of RDBMS) and connect to that using their computer through the internet. He will provide the infrastructure but charges for the same. Users used to pay RDBMS license.

1990 – Data Warehouse by Larry Ellison. This is just an extension of RDBMS. Store less used data in Datawarehouse and highly used data in RDDMS. Whenever required bring the data from the warehouse to RDBMS.

Data Analytics becoming costlier.

1995 – DOUG CUTTINGS came forward and stated he has an affordable solution. He introduced DFS (Distributed File System)

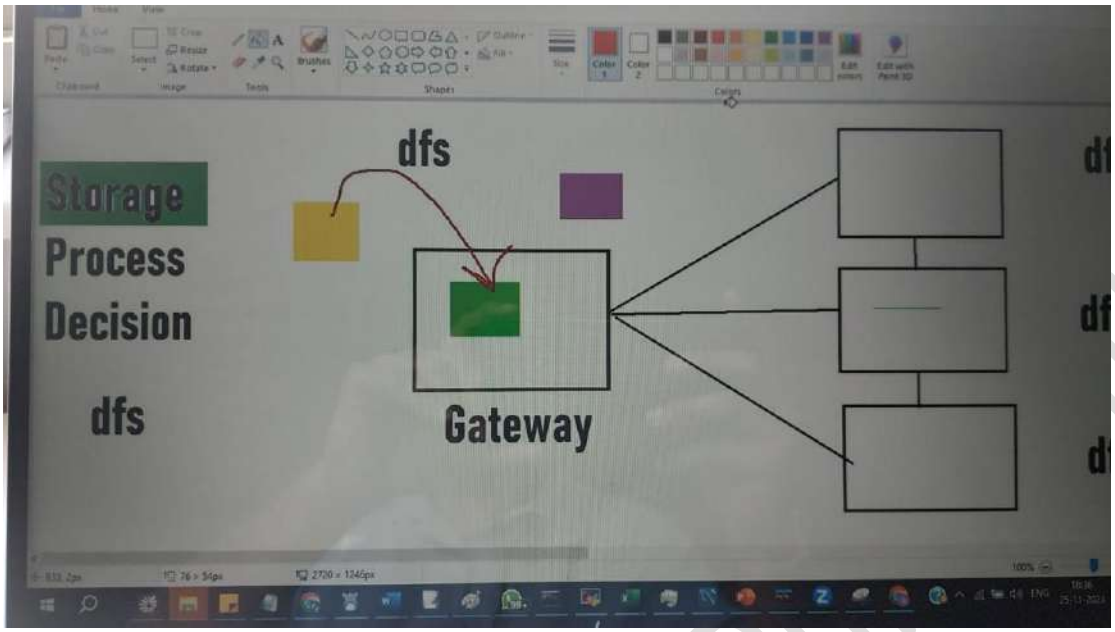


Storing

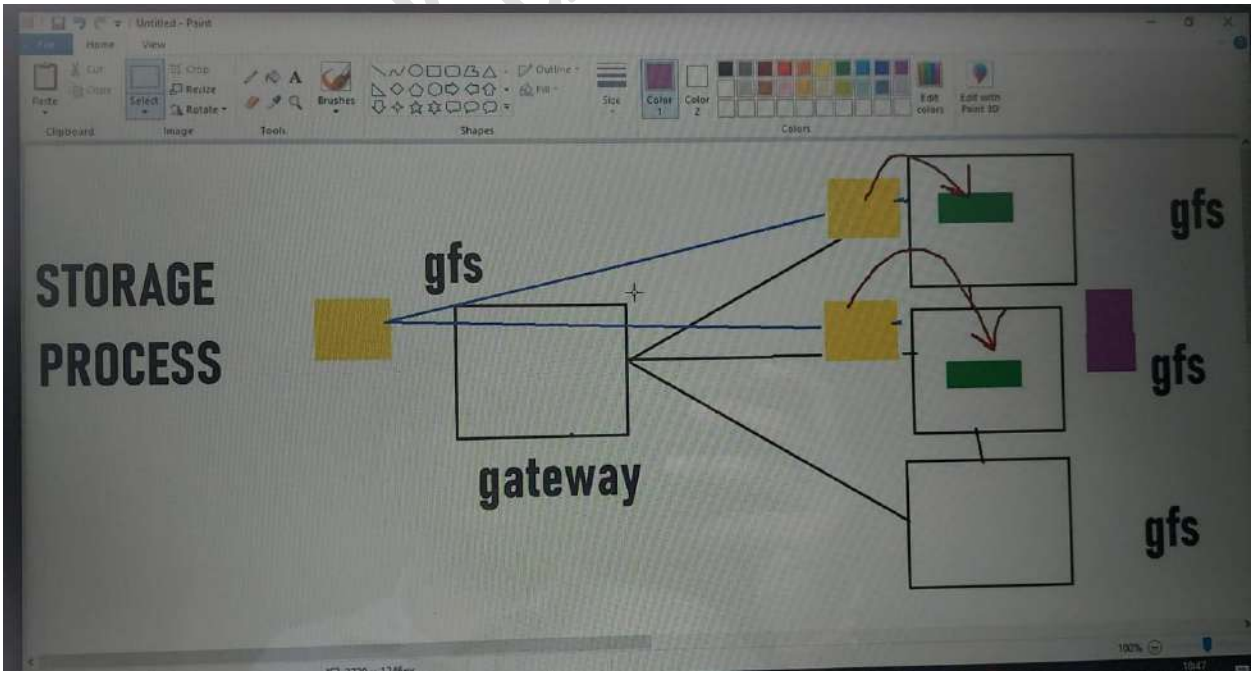
Arranging 4 laptops and Using one as a gateway laptop and rest as data laptops (nodes) and interconnecting them, When Data comes to Gateway laptop they will be distributed to data laptops. dfs is a software that he install in all the laptops to make this work.

Processing Query: When processing request is triggred, data from data nodes comes back to gateway and process gets applied and generates results.

When Data Brough back to gateway, the gateway laptop capacity should be very high, which was not accepted a good solution and this became a failure.



2003 – Google released a paper named as GFS (google file system)
GFS is same as DFS
2004 – Map reduce (Google processing Paper)
Doug Cutting used this paper to improve his processing
ProcessQueries will be sent to the data nodes and processed in the data notes and results be consolidated from all the data nodes.



2005 – Doug Cutting, looking at map reduce paper, Created Map reduce compatibility for DFS.
2006 – Hadoop was introduced by Doug Cutting. Hadoop = DFS (Storing) + Map Reduce (Processing)

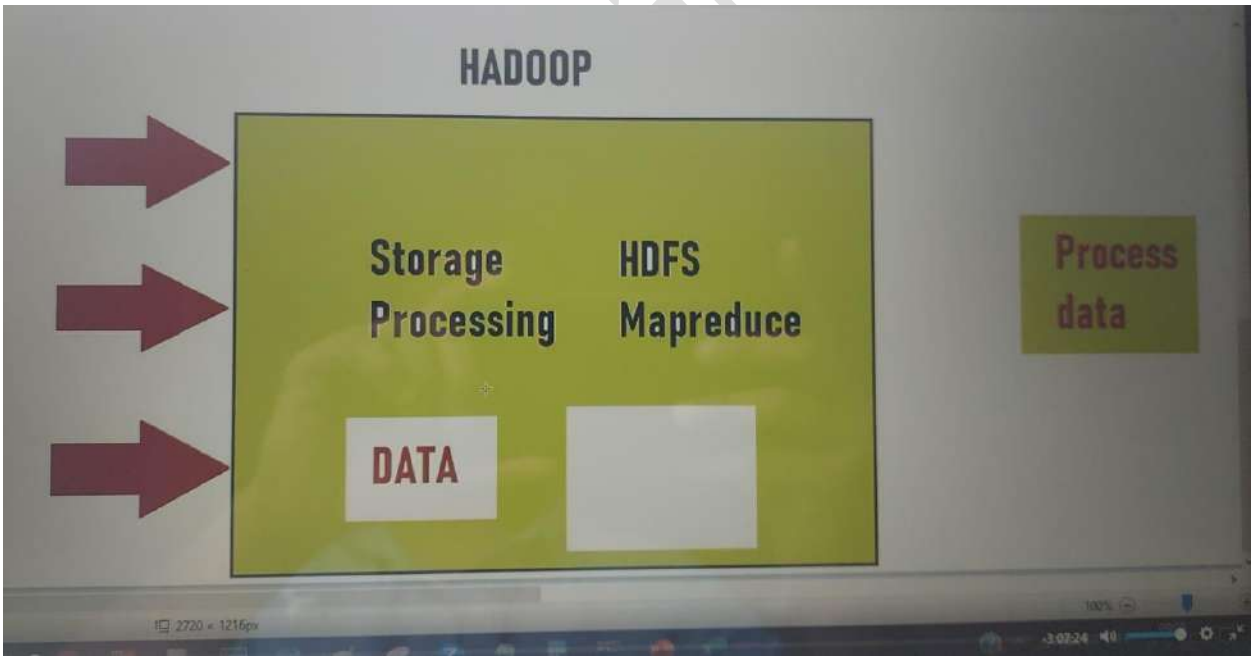
Big Data Execution and Learning Plan Discussion

Agenda - Day 2:

Simple Architecture of BigData

In Hadoop – Storage is called HDFS, and Processing using MapReduce.
Companies started dumping the data into Hadoop, processing the data, and taking out the processed data.

Get the data to Hadoop – process the data – take the processed data out



UseCase Execution – Basic

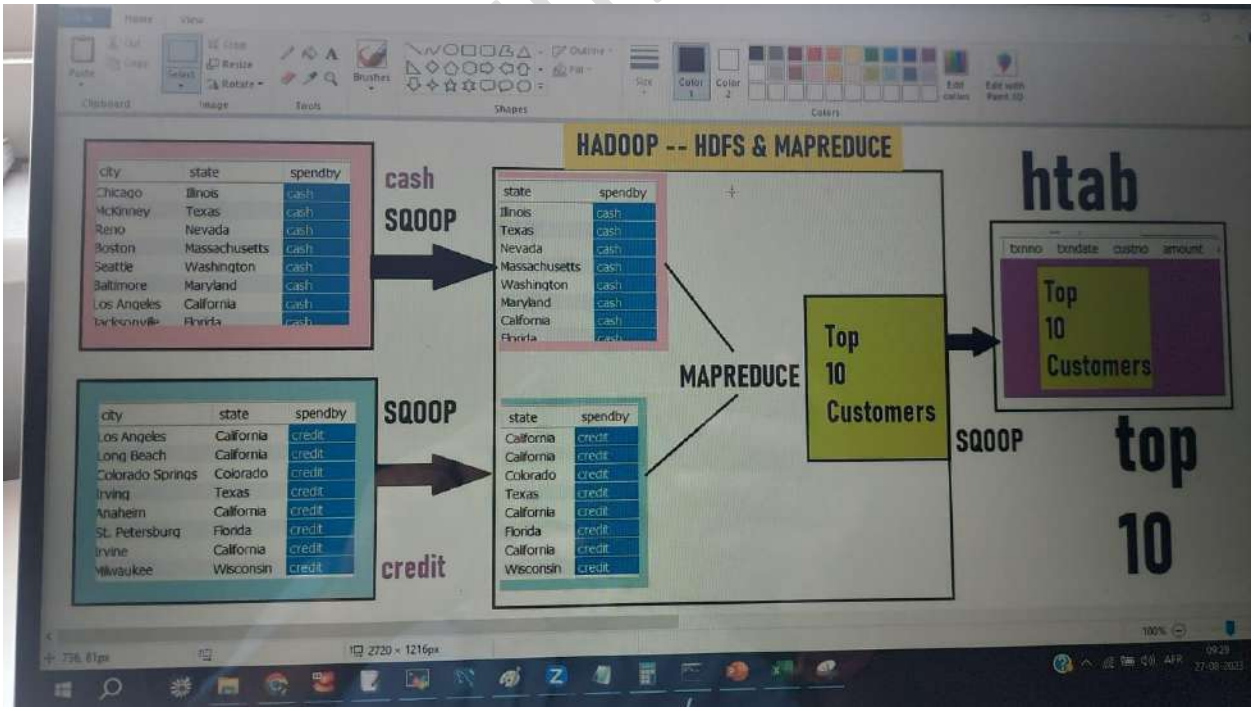
Walmart Problem – Planning to Analyze my entire sales data for last financial year, pick the top 10 customers, and release very good offers to them. But the data is huge, and seems to be costly, do any one can help?

Table 1 – Cash Customers
Table 2 – Credit card Customers

Requirement: Take data from above tables and store top 10 customers to Htab table

Solution 1:
Steps:

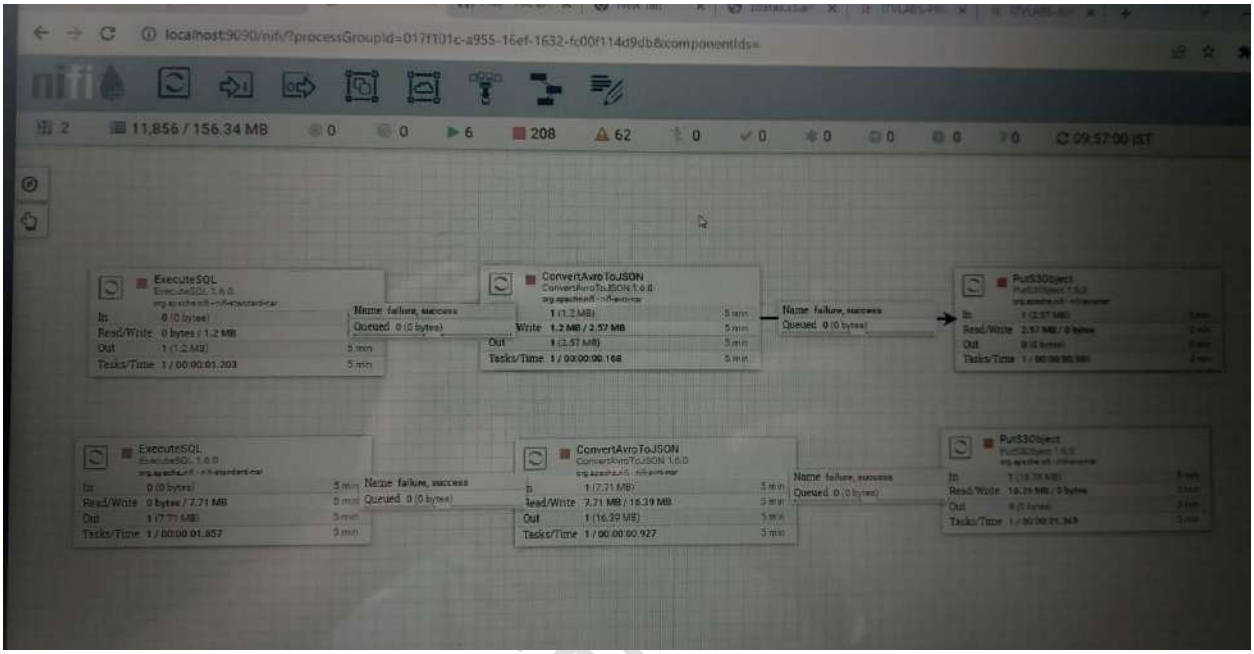
1) Place Hadoop system in between source and destination
2) Extract the cash data and credit data to HDFS using a tool known as Sqoop.
3) Use Mapreduce to fetch top 10 Customers.
4) Use Sqoop tool to write to the Htab table.



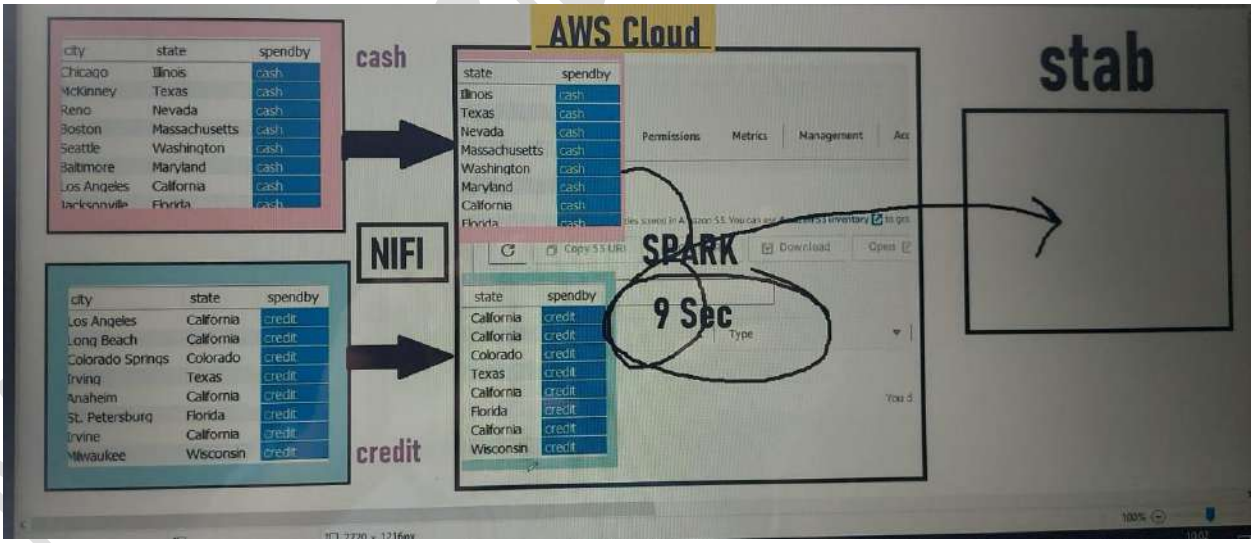
Solution 2: Much faster using AWS Cloud

Steps:

1) Add AWS Environment in between Source and Destination
2) Extract Cash Data and Credit Data with a Very powerful tool known as NIFI



3) Use Spark to Process the data



Setup Details (Configuration)
Downloads

Windows: 7Zip, virtual Box 6, Virtual Box 7, Putty, mobaxterm, cloudera
Mac: Virtual Box 6 , Virtual Box 7, Cloudera

Software Overview Hadoop deep evolution cloud introduction

Agenda – Day 3

Software Overview:
Business Analyst – Dev team, Test Team – SIT Server (System integration testing server), UAT Server (User Acceptance Testing), Live server.

Data Introduction:
19% of total data is generated by mobile phones.

Big Data Evolution Deep (Cloud)

2006 – Hadoop Introduced.

2007 – Issue with Ingesting the data to Hadoop. Solution: Sqoop is introduced.
2008 – Mapreduce found to be tough to implement/use. Solution: Hive is Introduced. Hive is just like SQL.(At the backend Hive implements Madpreduce)

2009 – Installation, maintenance and Upgrades became very challenging.
Have to take a laptop, Download softwares(Hadoop, Hive, Sqoop, oozie), Install Softwares, Maintain the softwares, upgrade the softwares.

Solution: Cloudera came forward for doing all the above-mentioned tasks at a cost

2011 – Hortonworks Came forward and said installation for free. But Pay me for Maintenance and Upgrades.

2014 – SPARK.
2018 – Cloudera Brought Hortonworks.
2023 – No Cloudera, Hartonworks. Cloud Computing is all that is live.(eg: Amazon EMR Service – Hadoop Cluster could be done -in less than few min)

Hadoop Clustering HDFS Architecture Cluster Admin Setup Details

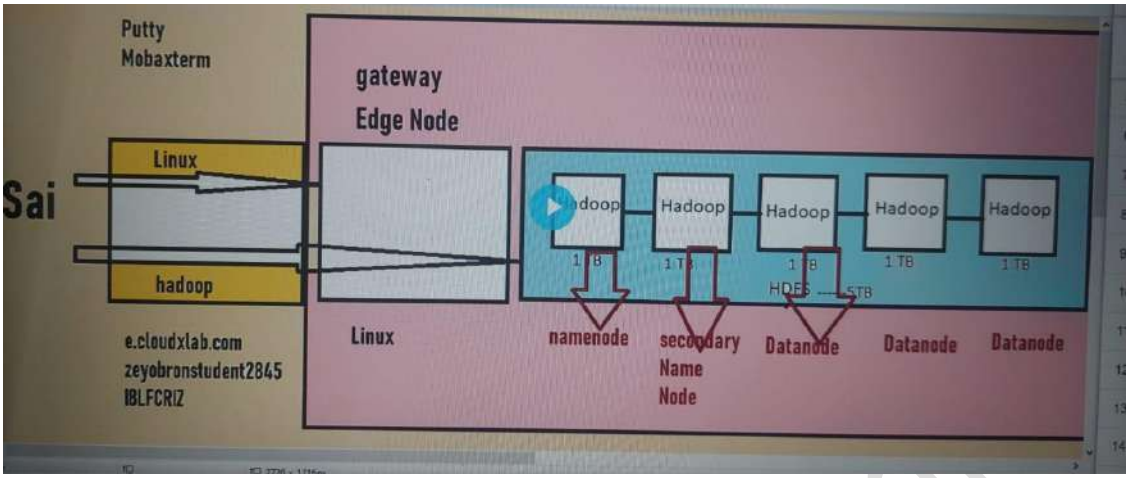
Agenda Day-4

Clustering

How do we communicate with the Hadoop Cluster and set it up?
Cluster – Group of laptops Combined.

Steps to Create and communicate with the cluster:

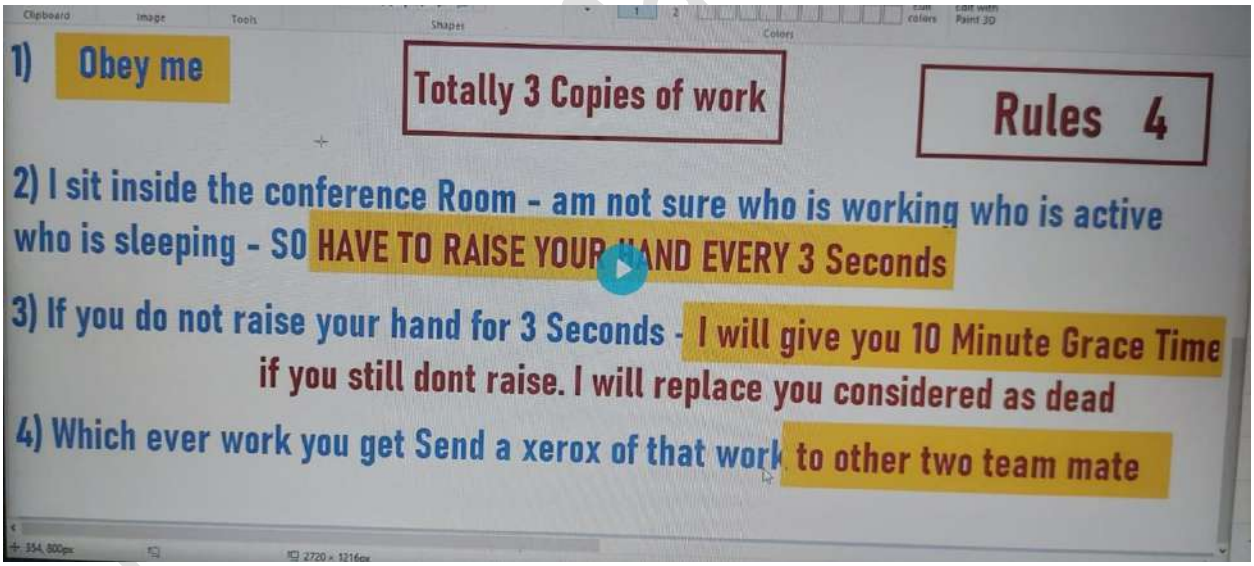
1) Take the Laptops required to form the cluster. (5 Laptops with 1TB Capacity)
2) Install Hadoop in all the Laptops.
3) Interconnect all the laptops, this would become a Cluster of HDFS (5TB HDFS)
4) Configure Gateway laptop (Edge Node) to talk to HDFS Cluster
5) Use Edge Node(Gateway laptop) IPAddress/Hostname (hostname in cmd will give us hostname, ipconfig in cmd will give us IP address) and then use software's like Putty/mobaxterm to connect to it with valid credentials.
6) To Communicate with the cluster, we need to communicate through gateway node using Hadoop language. To communicate with the gateway node, we need to use Linux Language.
   Eg: linux Command: ls
   Hadoop Command: Hadoop fs ls
7) Names of the computers in clusters- Name Node, Secondary Name Node, Data Node

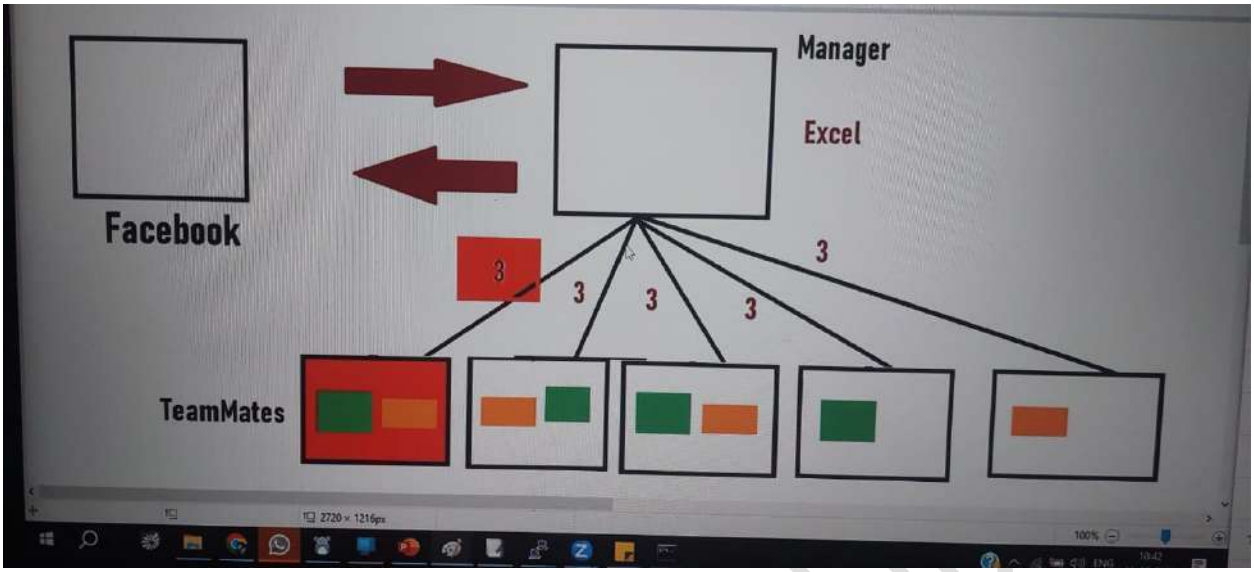


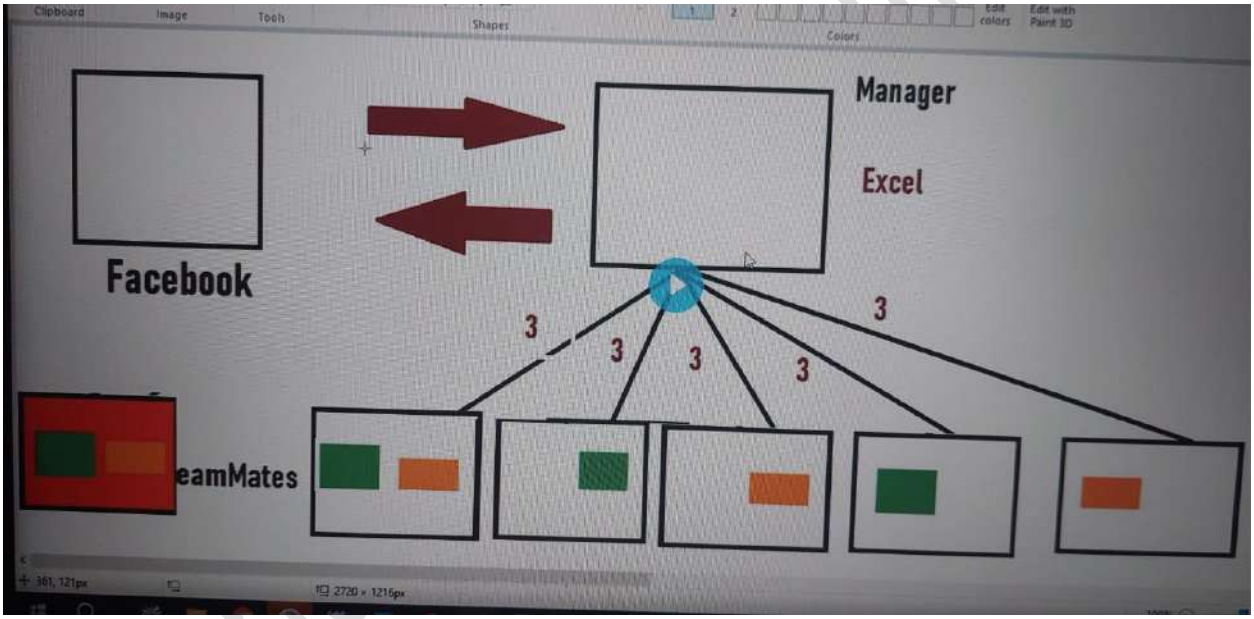
Test Cluster Details:
Hostname: e.cloudxlab.com
Username: zeyobronstudent2845
Password: I8LFCRIZ

A Short Corporate Story

Facebook – TCS Work contract Story.

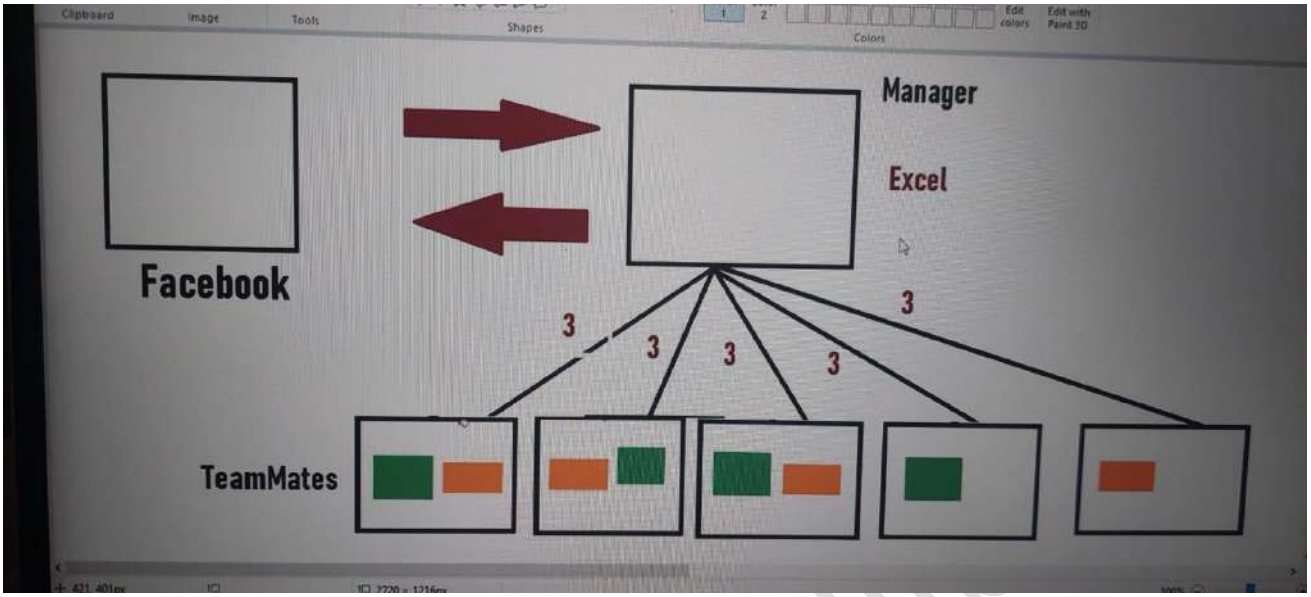




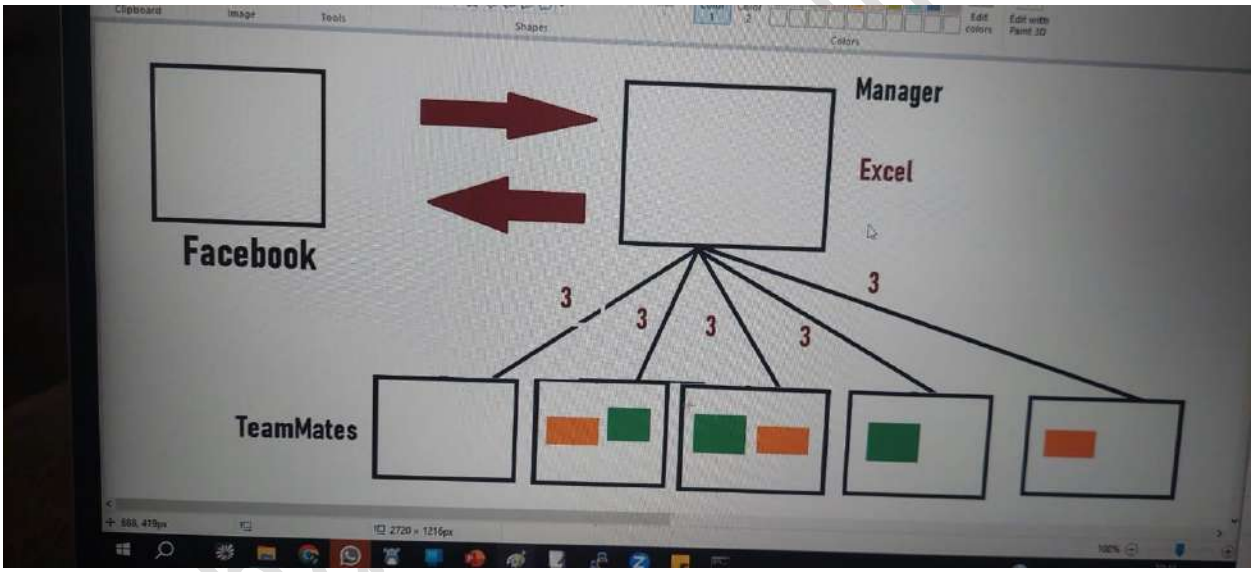


When a node is not responding back for 10min, the Manager redirects the other nodes to replicate the data to 3 copies and asks resource manager to replace the dead node. If in case if the node which is considered to be dead and about to replaced started responding, then manager asks that node to delete all the copies it has so that manager rearranges the workload again to 3 copies (replicas decided).

When dead node started responding back – we get 4 replicas

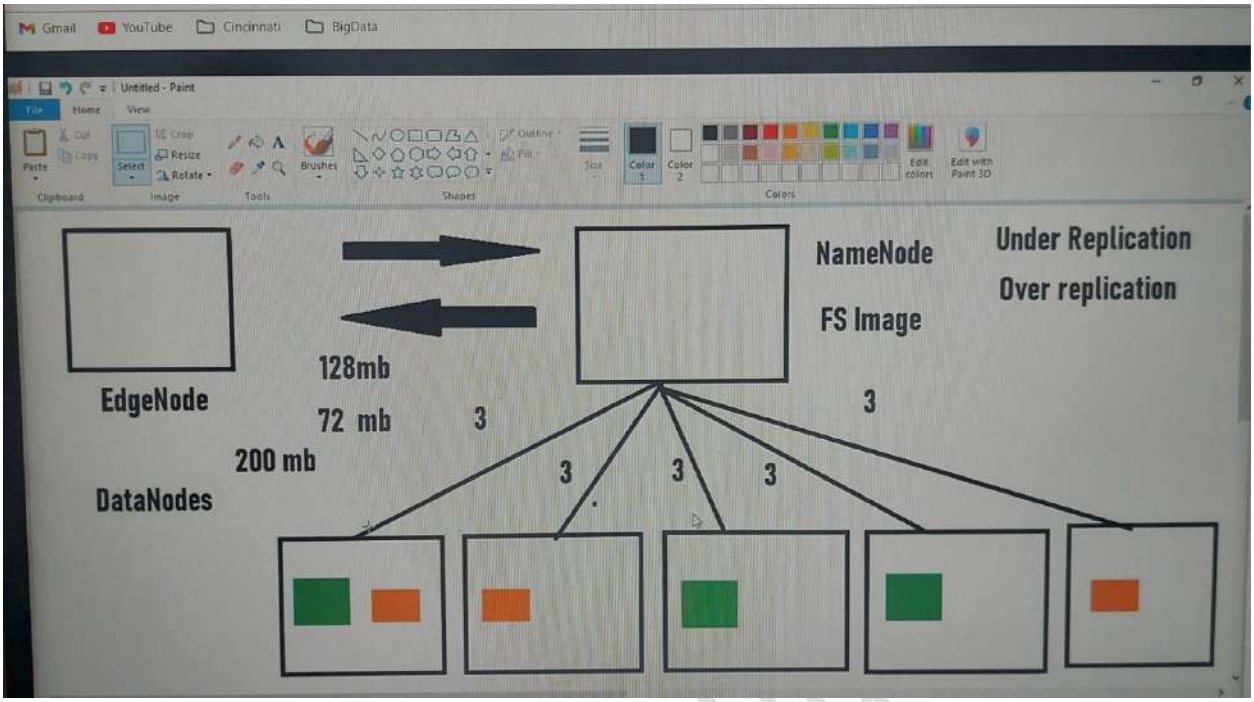


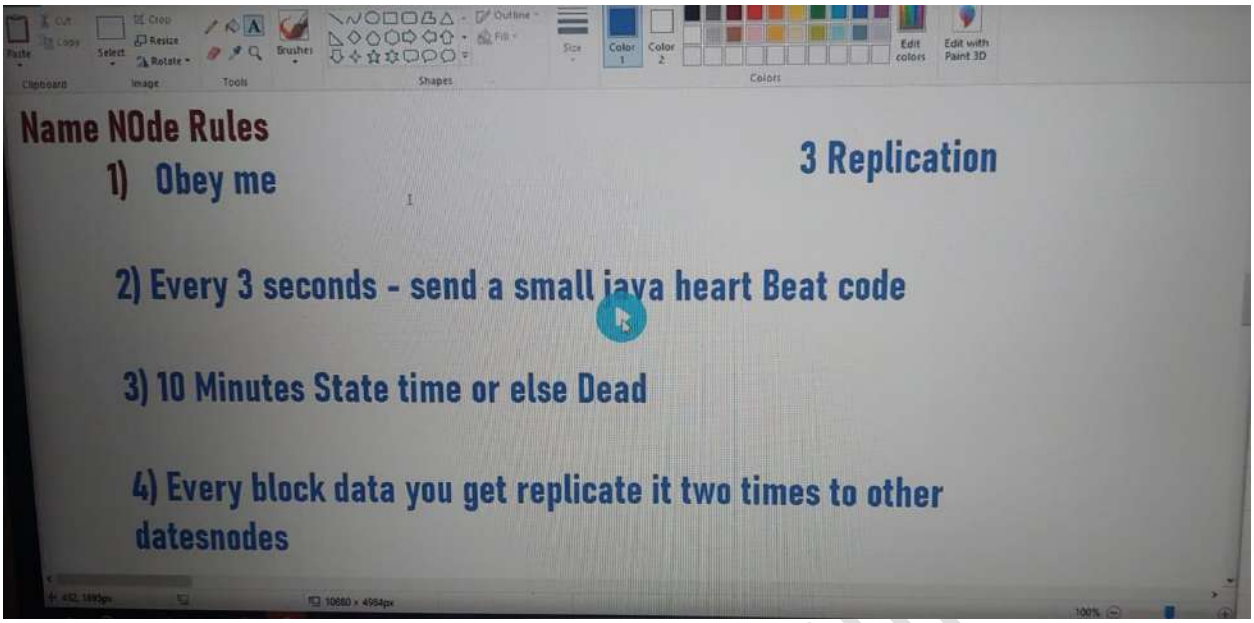
Now manager asks to delete the copies of the node that started responding back.

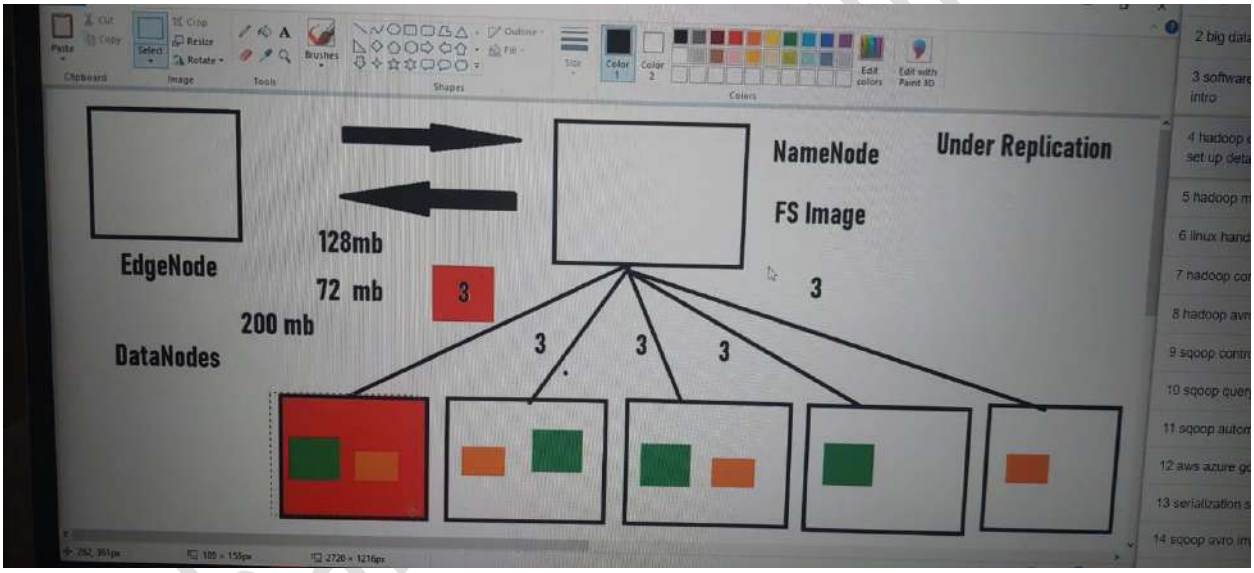


HDFS Architecture

Mapping the above story with the HDFS Architecture., with the actual naming conventions







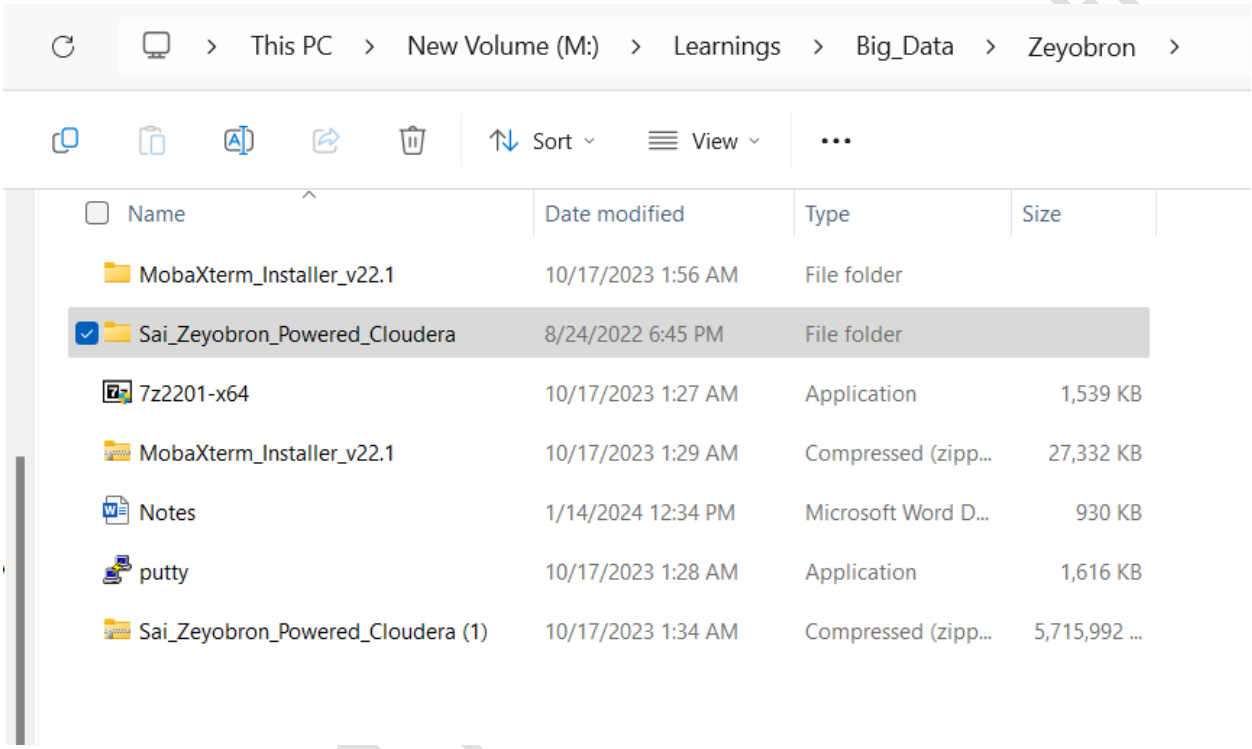
When the dead node becomes active, it becomes over replication.

Practical Block Splits

Practically Showing the above DataSplits

Setup

1) Install 7z
2) Install Virtual Box
3) Extract Cloudera Using 7z.
4) Pull the extracted Cloudera to virtual Box.



Video Reference: https://www.youtube.com/watch?v=u1x0ZMEeefU

Hadoop Metadata Management Hadoop 2.0

Agenda:

Known facts:

1) The client (Edge Node) is important to the Manager (Name Node), and responds faster.
2) If any file/app sits in RAM, it responds faster. All the applications will be in HardDisk, when an app is started it comes to RAM and when we clear Apps and Cache it goes back to HardDisk.

3) If the file size is huge. To save to HDD – it takes time.
4) Risk – All of a sudden if the laptop/mobile restarts – RAM will be flushed away.

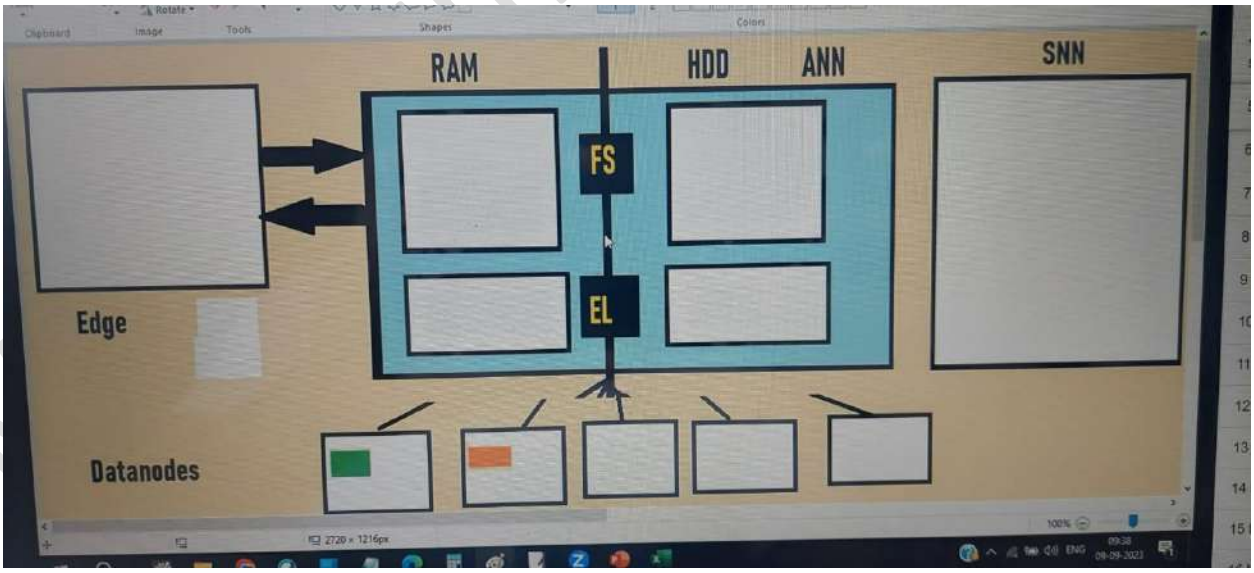
Hadoop metadata Management

fsImage is the file that will have all the metadata information and will be used by the name node to respond to the edge node. It's good to have FSIamge in the Name node RAM. But, if the name node restarts we will lose the FSImage file. And also it's not a good idea to store FSImage to harddisk as it's a huge file, the system may hang once in a while which causes a delay in responding to the edge node. then what is the solution?

FSImage Handling is important – As it involves client's communication.

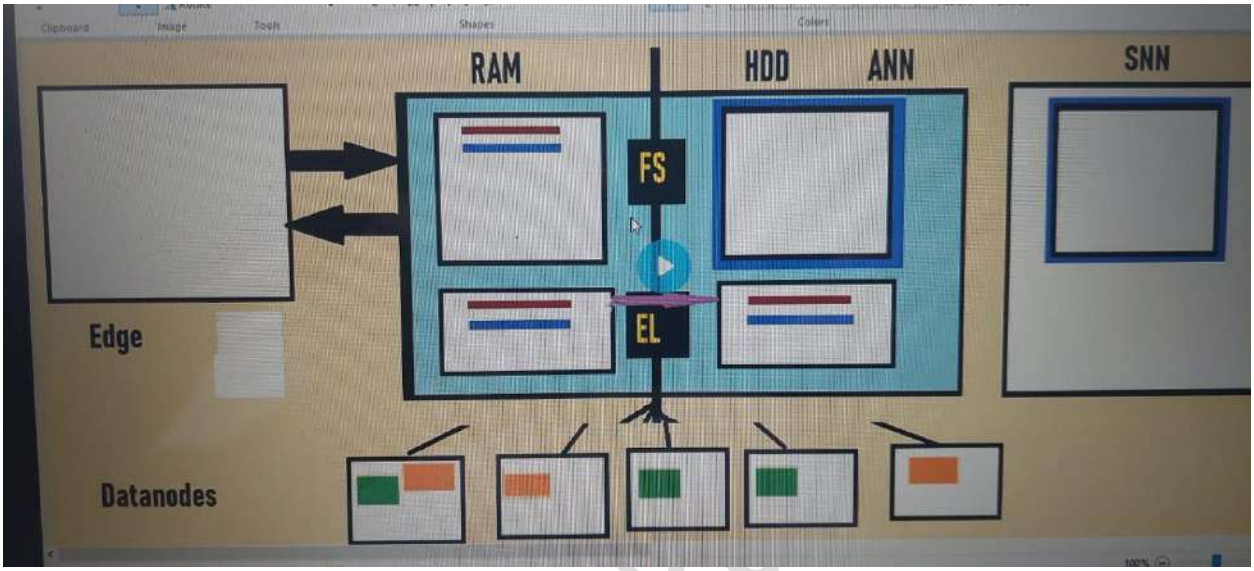
Below is the Architecture that solves the above problem.

1) Initially for any Hadoop cluster FS Image, Edit lock file will be in harddisk
2) Once Cluster is Started, it brings FSImage and Edit Lock to RAM

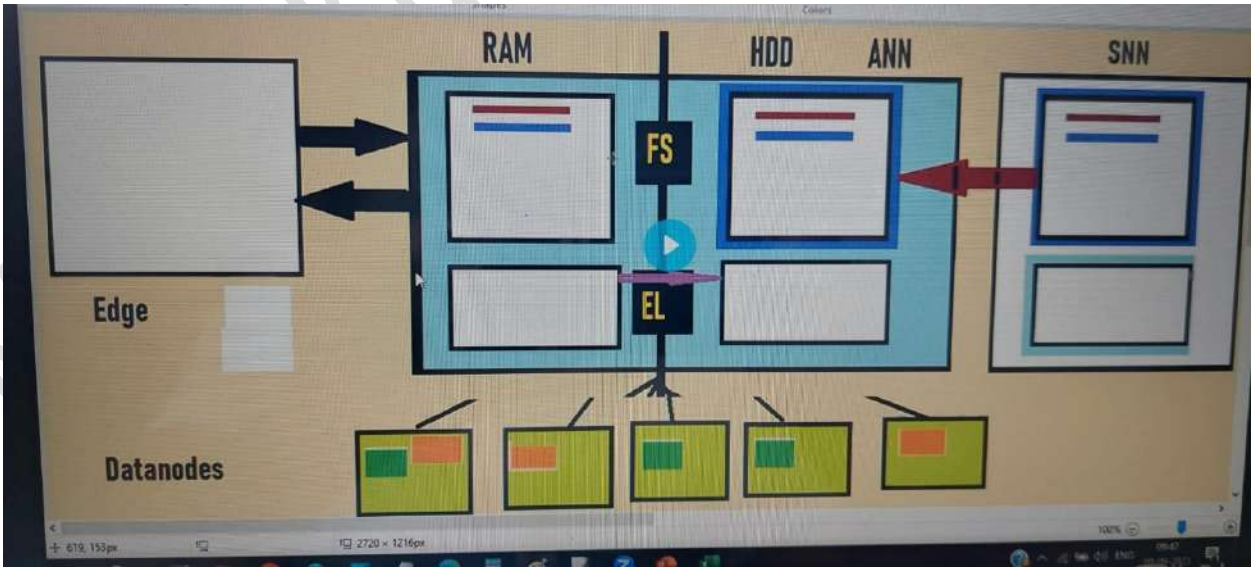


3) Any New Transaction (Data From Edge Node Stored to Data Nodes. Heart beat data, Over replication data….) will be logged in both the FSImage and

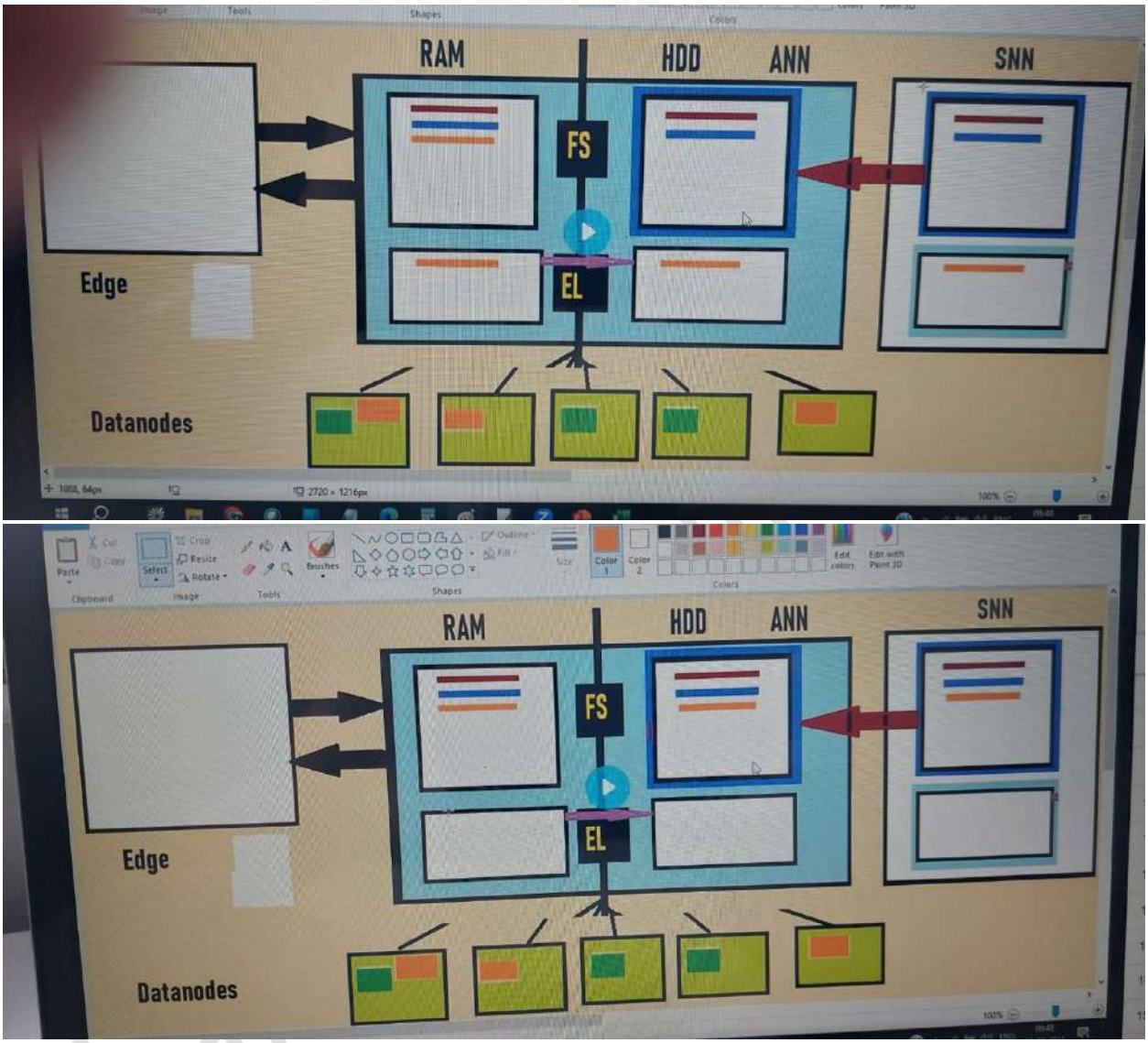
Edit_Lock File, Also, Edit_Lock will also push this data to Edit_Lock file in Harddisk. Now FSImage remains untouched.



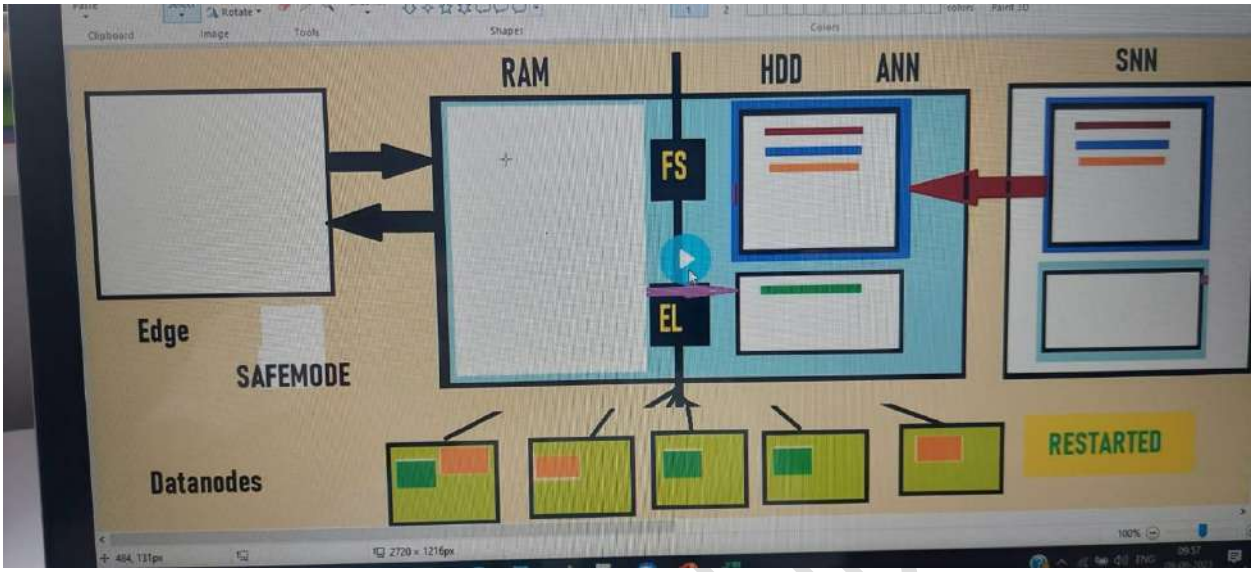
4) Now, for every 1hr name node send the FSImage in name node HardDisk to Secondary name node(SNN) and also updated edit lock file in name node ram to SNN. Then, Edit Lock file in Namenode Ram and HardDisk(HDD) will be flushed/ cleared. Edit Lock file in SNN will be merged with FSImage in SNN and pushed to Namenode HDD FSImage.



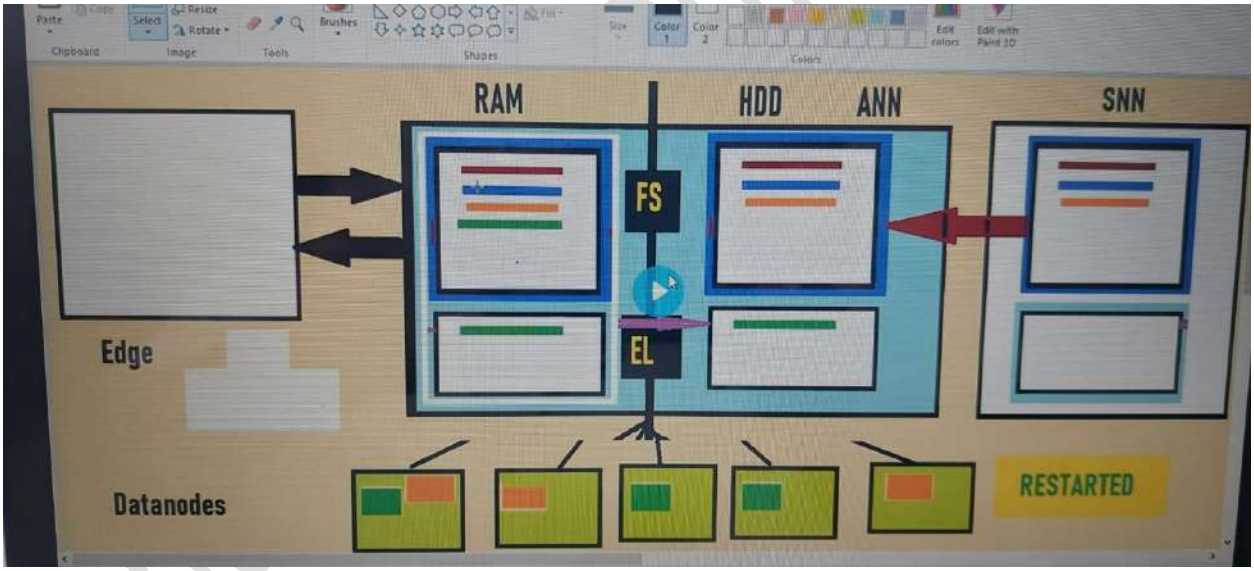
5) New Transaction



6) When Name rode restarted any time with in 1hr, before SNN process the merging,As all the data in name node wipes away with restart Name node goes to safe mode, In Safe Mode – name node gets the copy of FSImage and EditLock from Name node Hard Disk, Performs merge, Goes out of safe mode and Serves Edge Node. In Safe Node, it wouldn't serve any request.



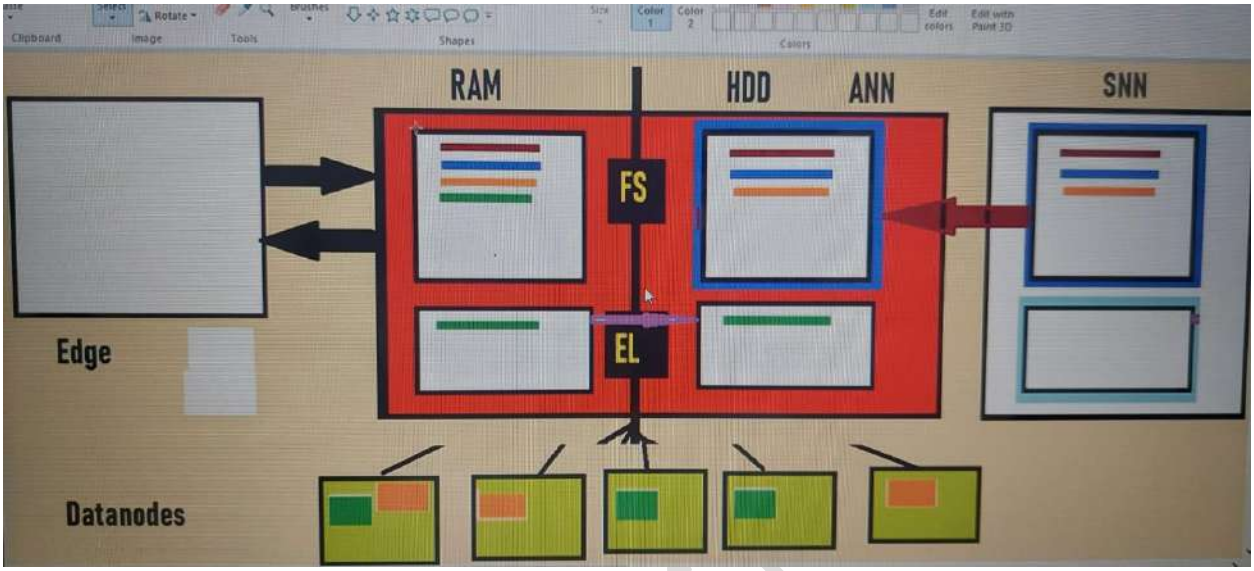
Auto-Merge and goes out of Safe Mode



What name node completely fails? It doesn't have a backup, The same is introduced in Hadoop 2.0

Hadoop 2.0

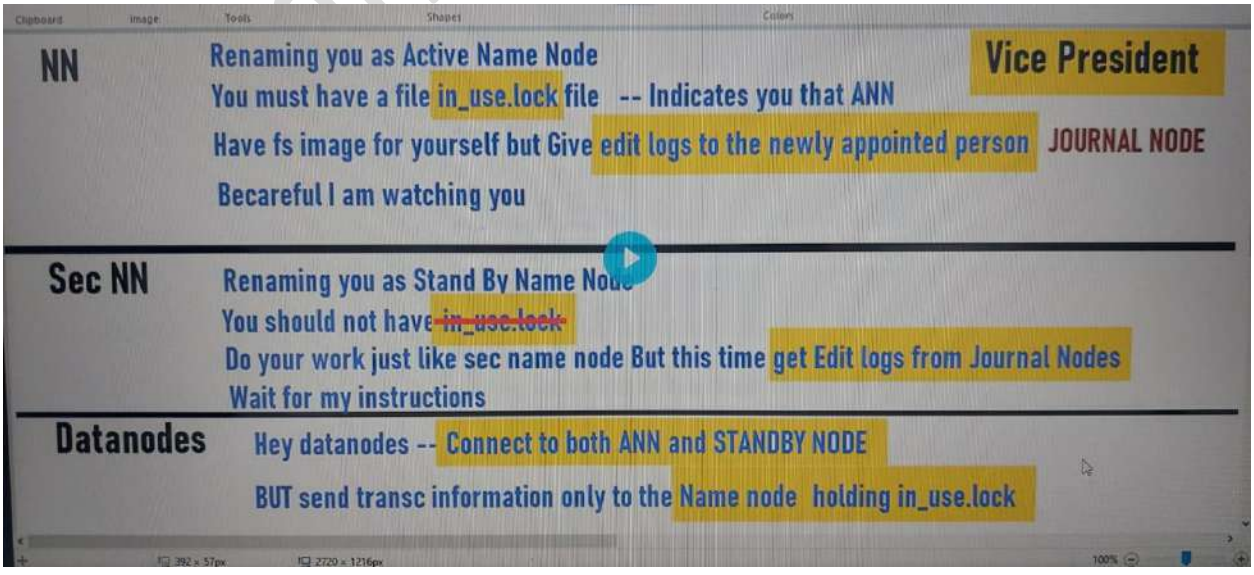
The name node completely went down. SNN can't be active and serve the requests because it doesn't have the latest transactional data.

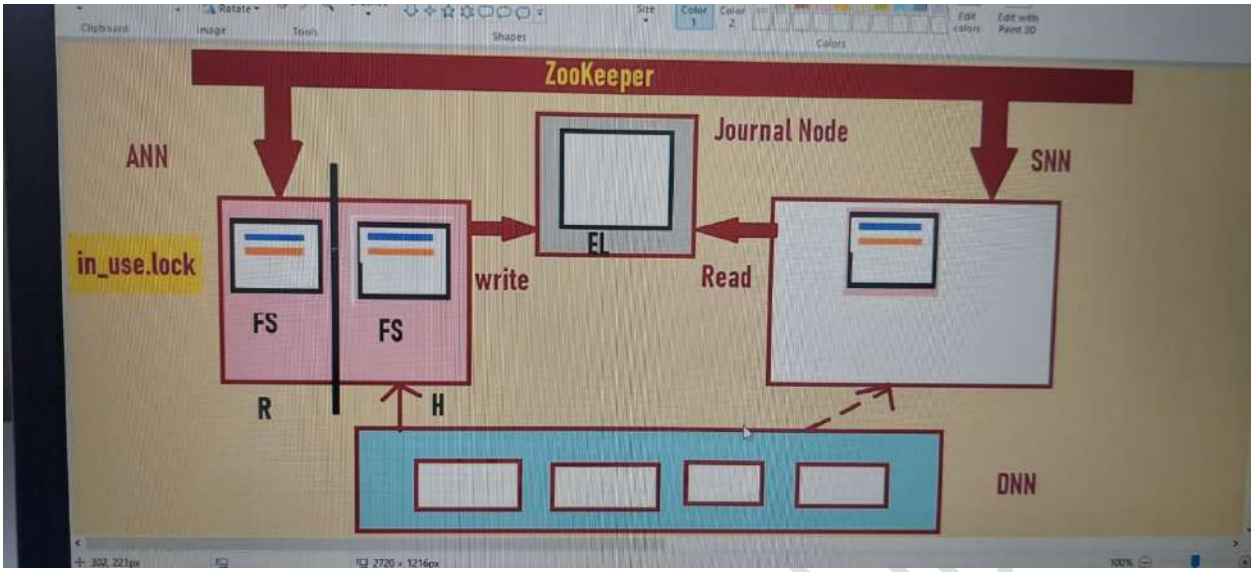


Single Point of Failure is the major issue with Hadoop 1.0.

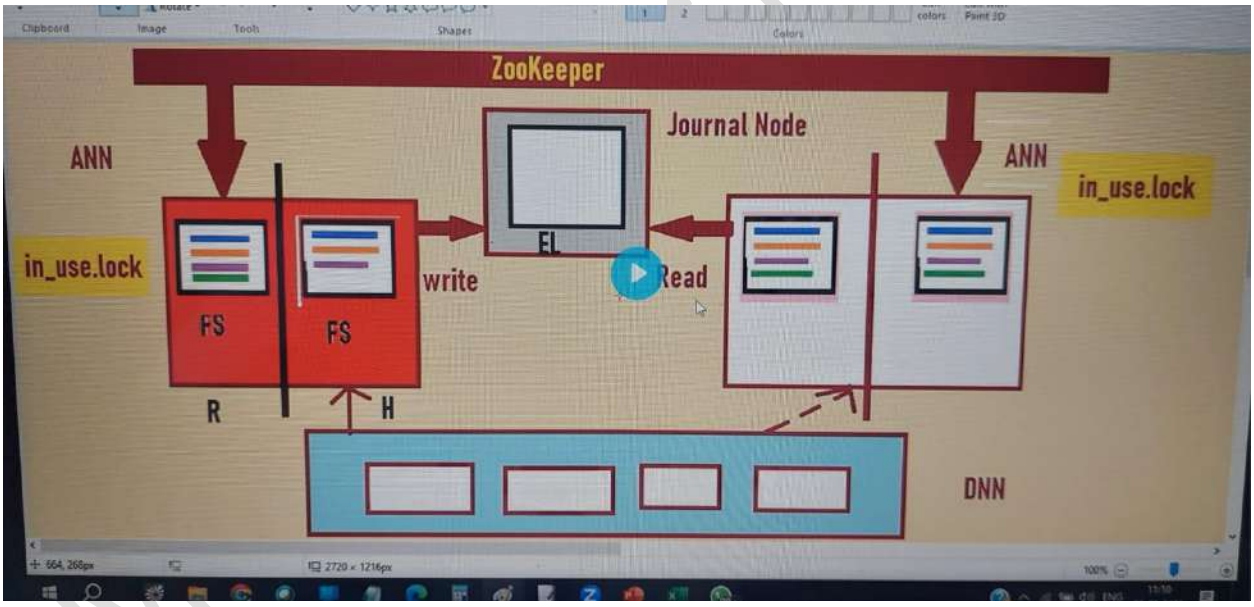
July-2013 Hadoop 2.0 got launched.

To Monitor Managers (Name Nodes), vice president (Zoo Keeper) are introduced in Hadoop 2.0

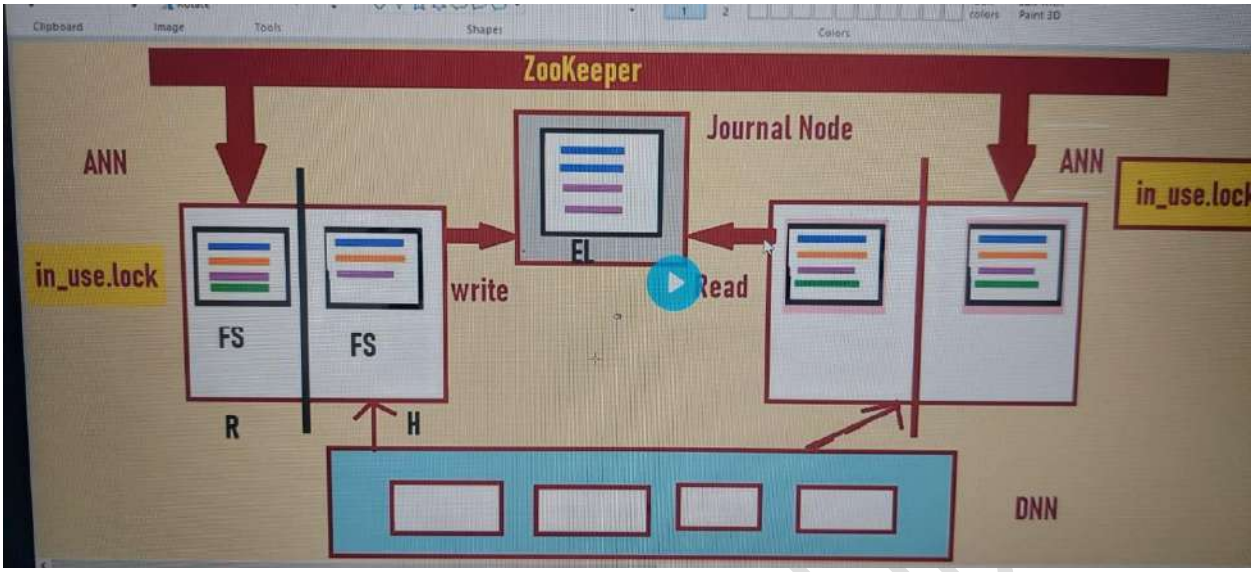




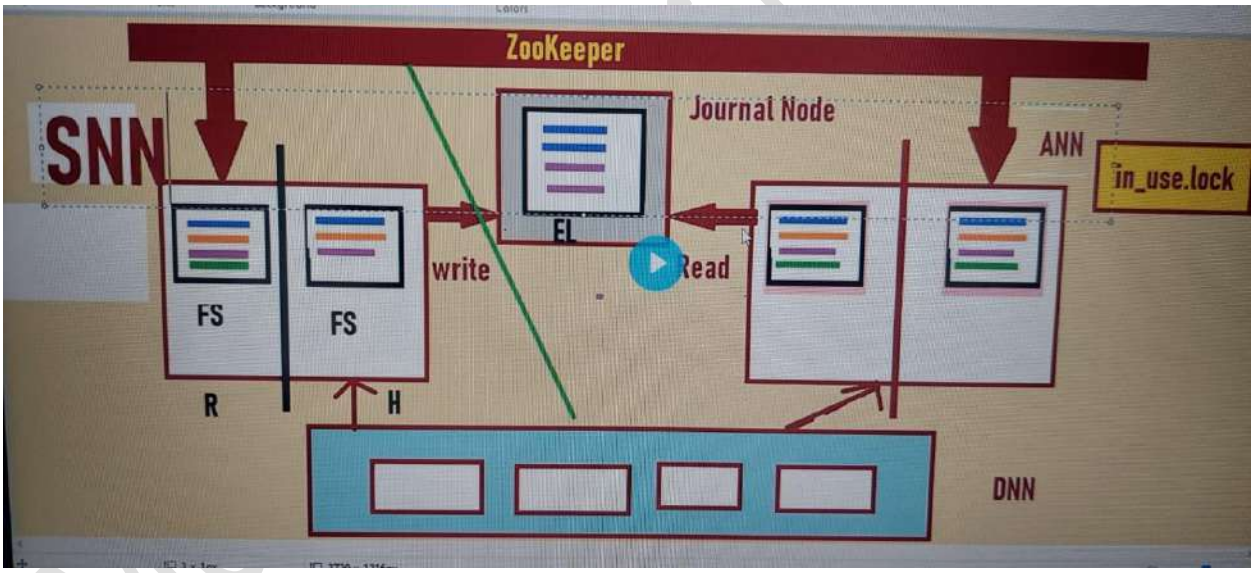
When the Active Name node goes down, Zookeeper Redirects the SNN to create its in_use.lock file and get the latest EL from the Journal node merge the Transactions and serve the edge node, SNN becomes ANN now.



If Dead ANN becomes Active, Then Data nodes will send the transactional data to both the nodes, as a result, we get duplicated in Edit Lock. This scenario is called Split Brain Scenario.



Now, Zookeeper immediately finds this and sends a small fencing java code to stop the ANN that became active from dead state and asks it to remove in_use.lock file from it and then it becomes SNN.

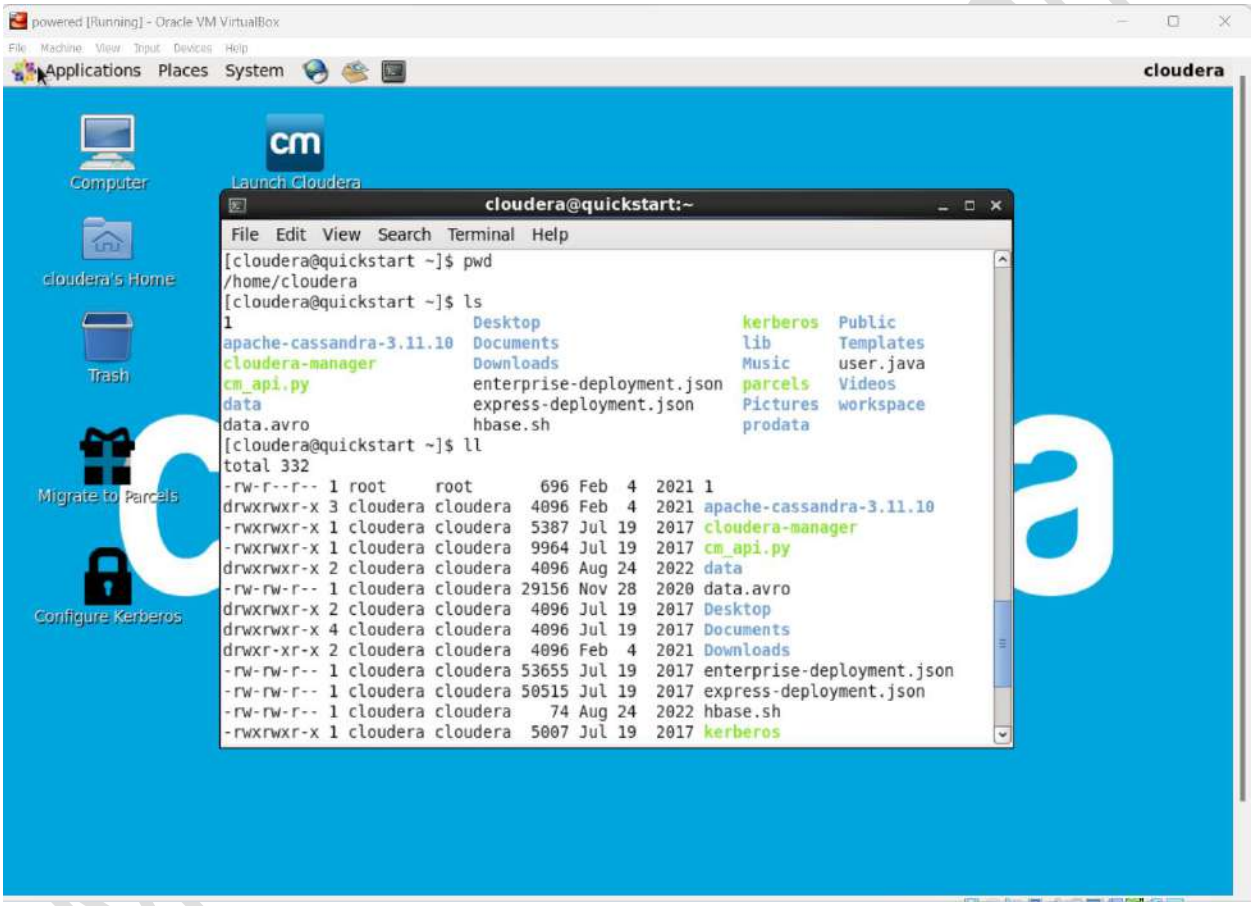


Linux Hands-on Hadoop Intro

Agenda Day 6

Linux Hands-on

pwd, ls, ll, clear, mkdir, touch,cd, cd ..
Relative Path, Absolute Paths in Linux Discussed



Linux is used to communicate with Edge Node

Hadoop Hands-on

Home Directory of the HDFS is /user/cloudera/

All the linux commands should be appended with “hadoop fs –“ for Hadoop commands

Home Directory of cloudera is /home/cloudera

Hadoop Command Deep Hands-on

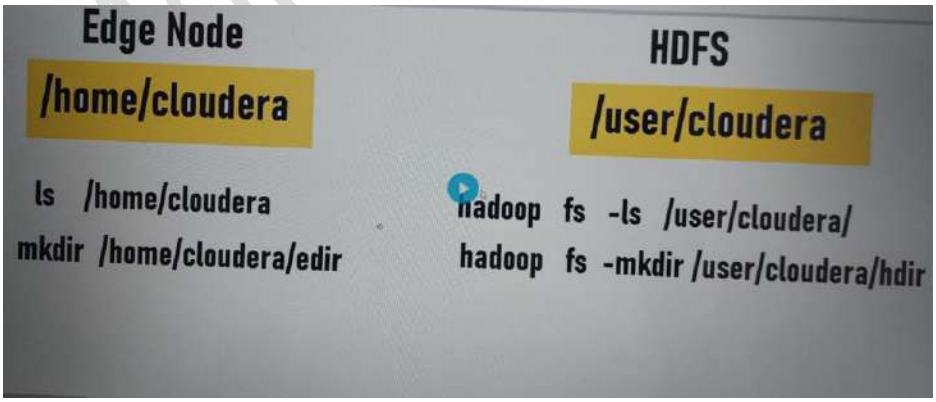
Agenda Day-7

Linux – echo and cat commands
HDFS Commands

Hadoop for big Data Engineers is a kind of Single File System. Handling replications, making sure Name node, Secondary Name node, Data node, journal nodes are up, and working is the responsibility of Big-Data Admins.

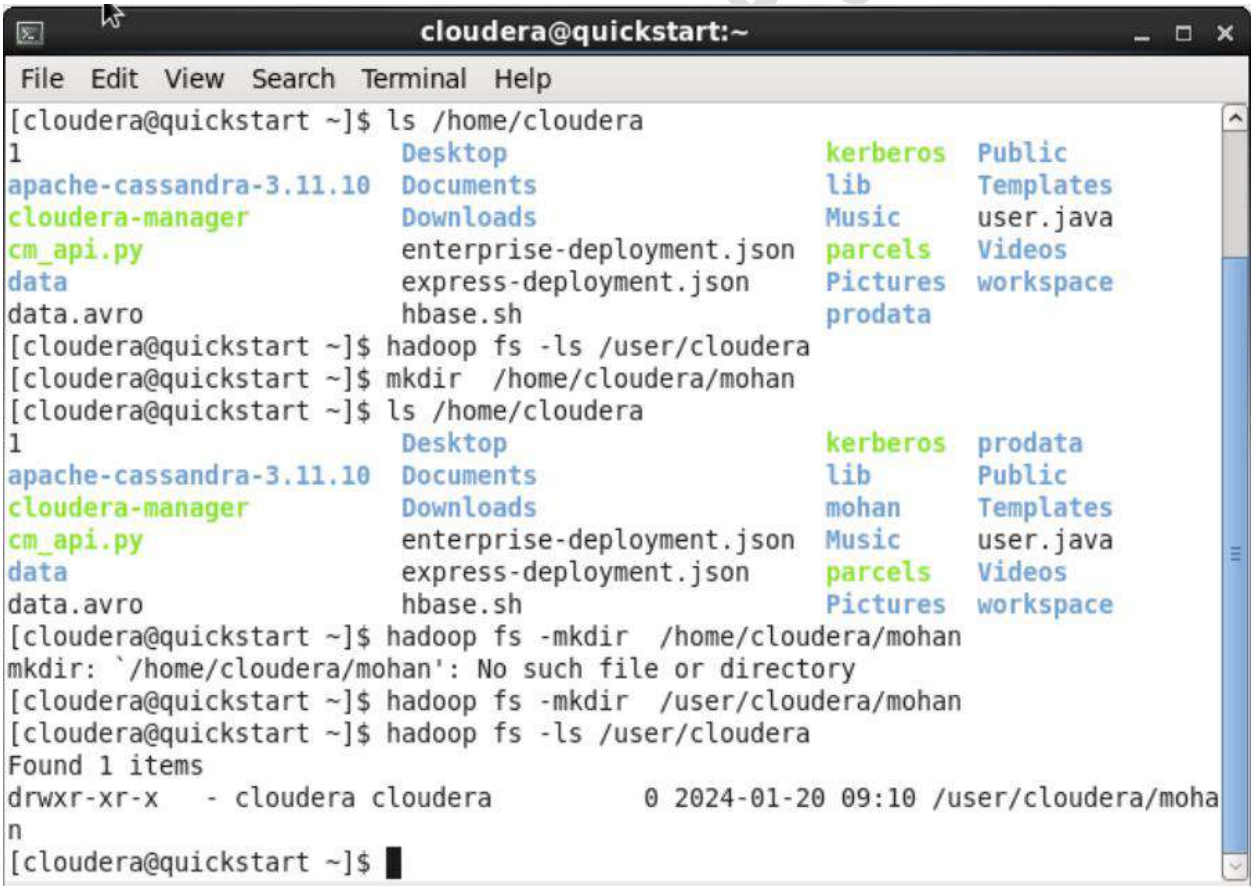
Sample Hadoop Commands:

hadoop fs -mkdir /user/cloudera/hdir
hadoop fs -ls /user/cloudera



Before starting any work, When Using cloudera, the below command should be the first command that always should be executed.

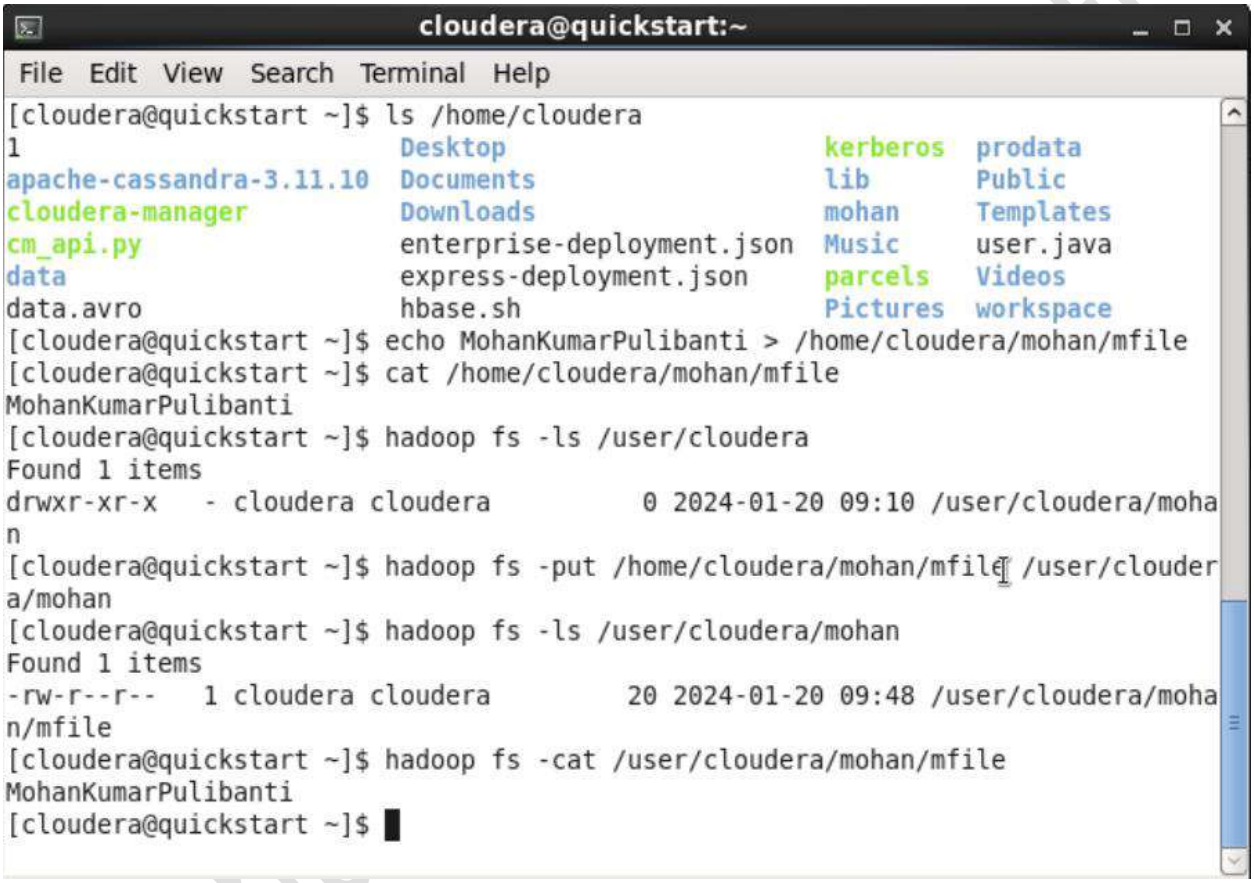
hadoop dfsadmin -safemode leave



Scenario:

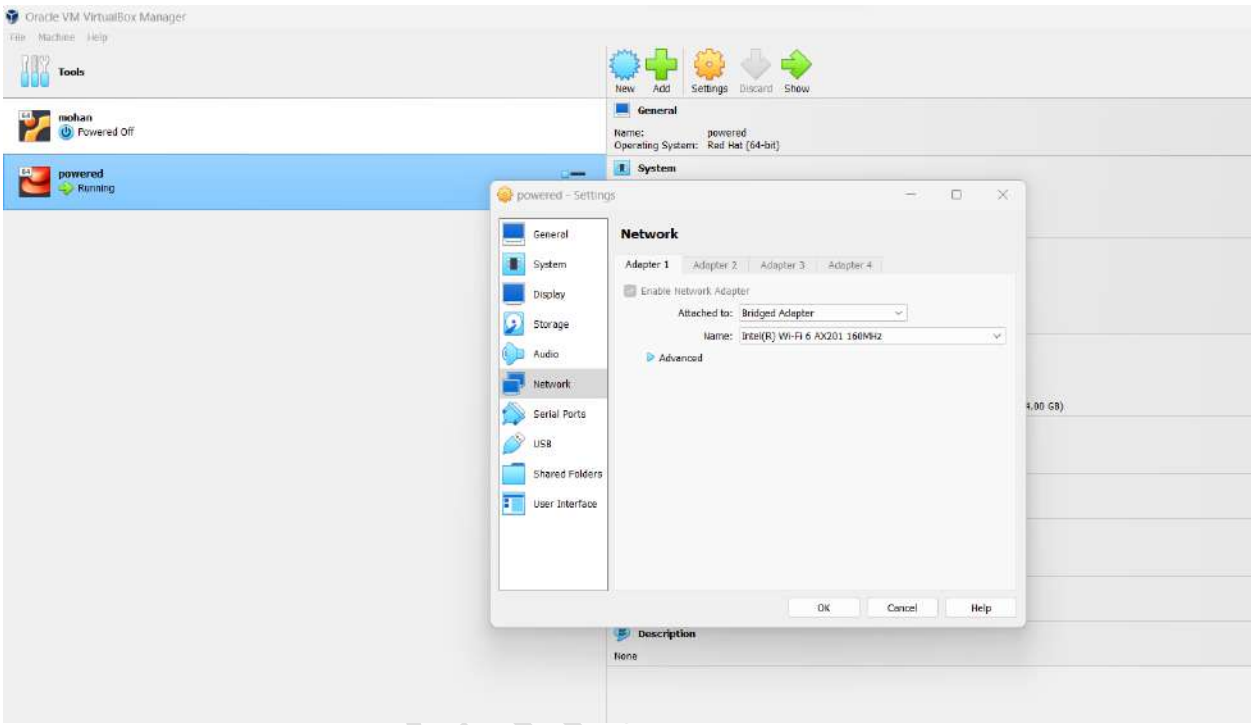
Create a file in Edge node and move that to hdfs.

Command to put the data to hdfs from Edge Node.
hadoop fs -put <EDGE_SOURCE_Path> <HDFS TARGET>

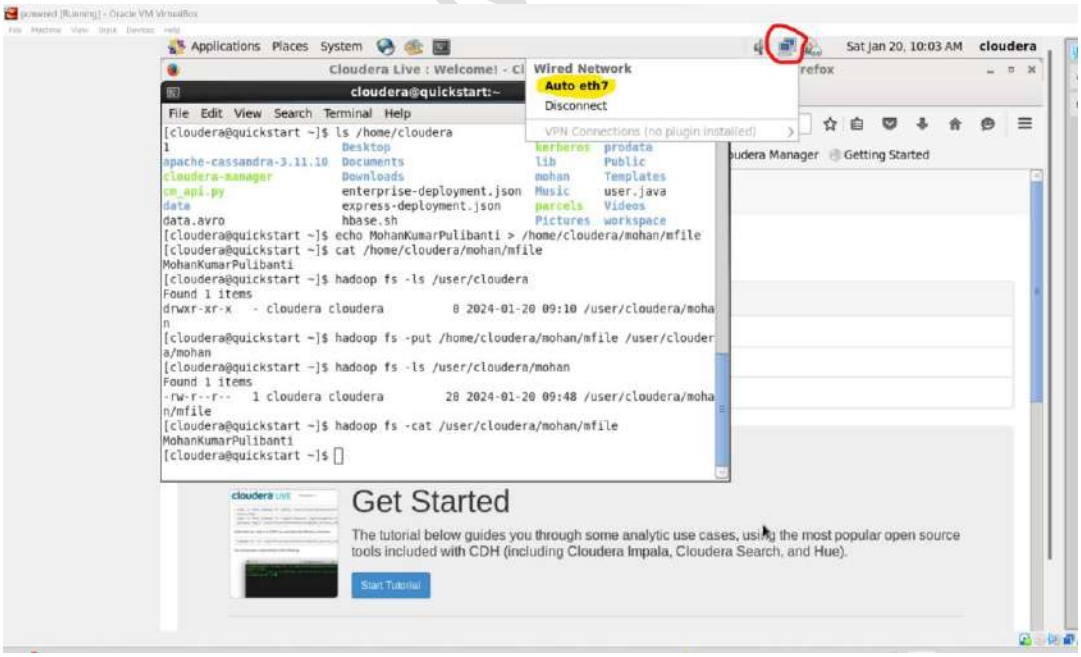


Configure Putty:

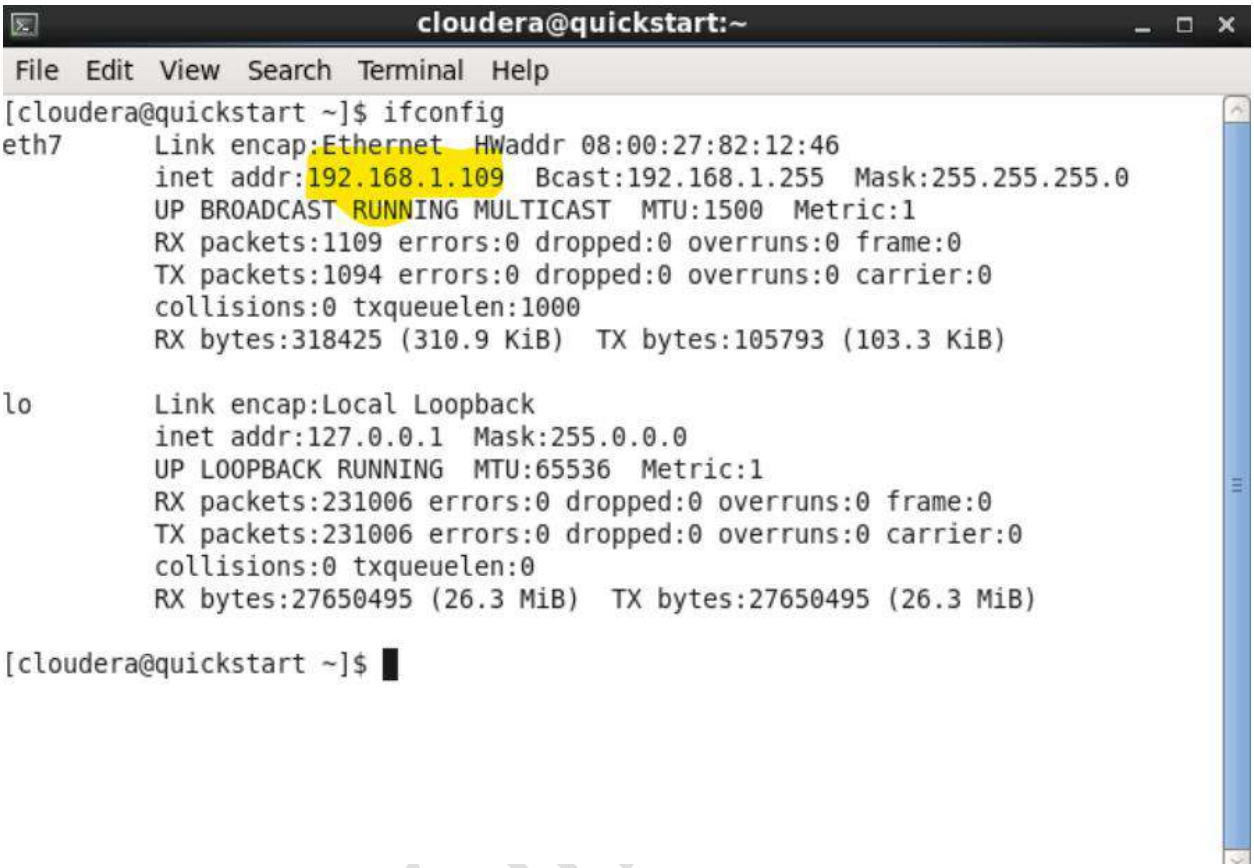
Open Oracle Virtual Box – Go to settings – Network – Choose Bridge Adapter in Attached To: dropdown – click ok



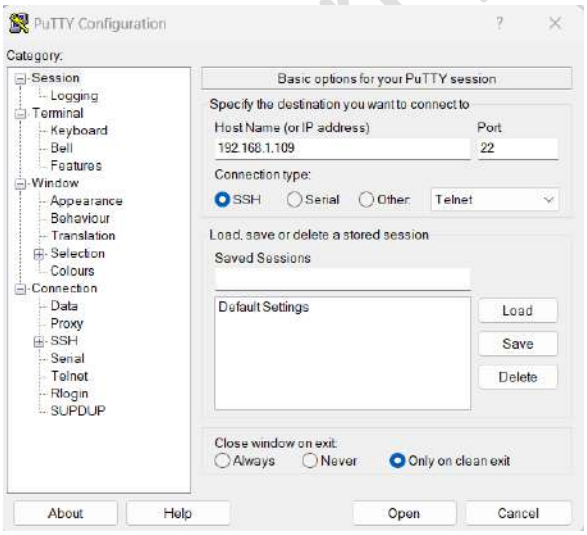
Now go back to Cloudera instance, on the top of screen we can see a double computer – click and select Auto eth7

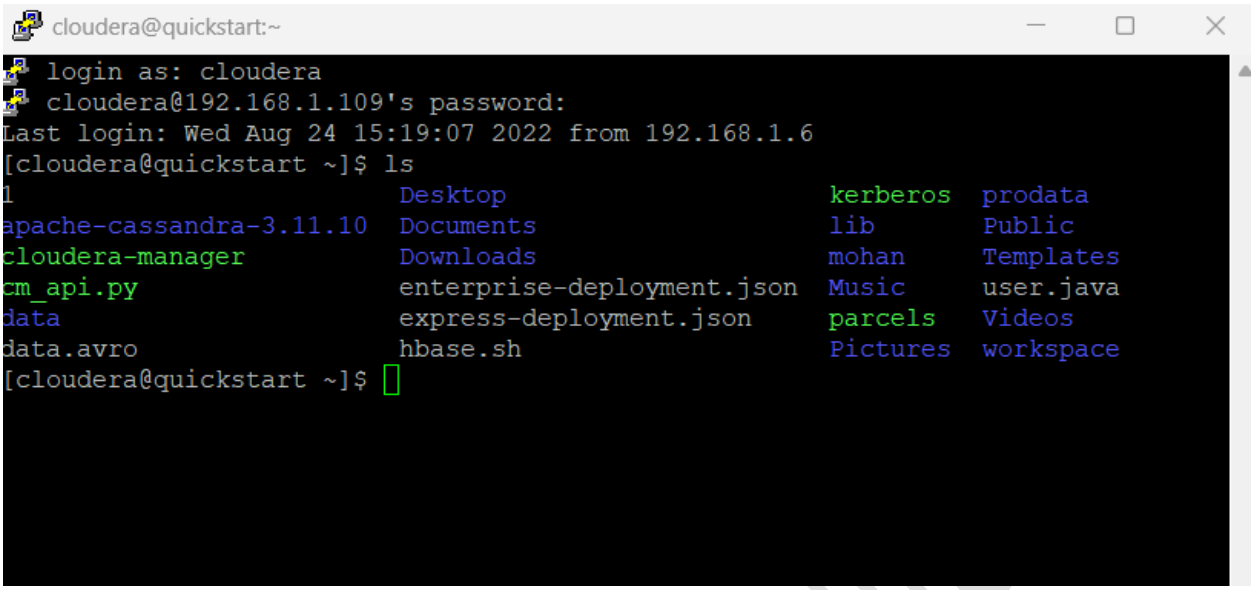


Open new terminal → and type ifconfig



Open Putty and use the above highlighted IP and use username & Password as Cloudera.





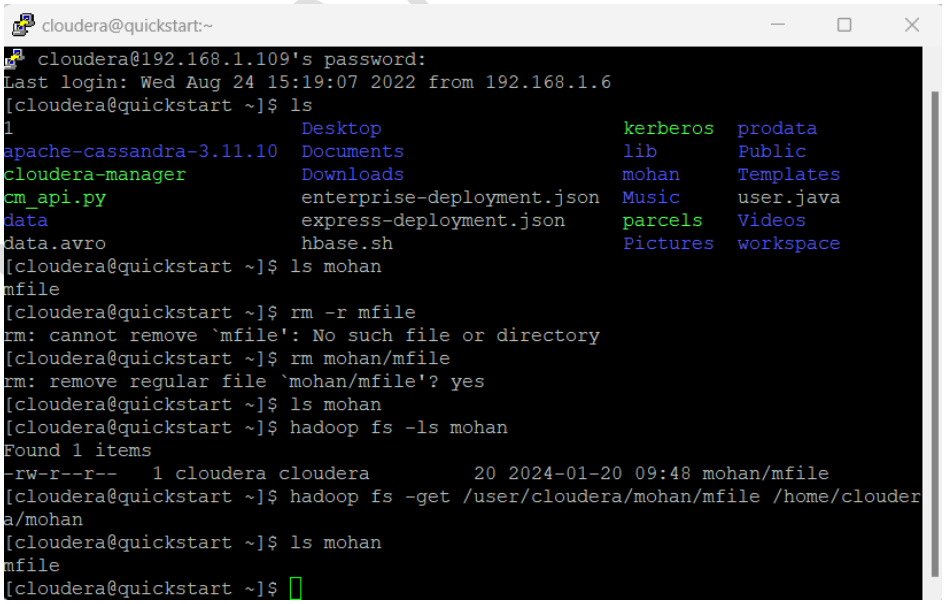
Hadoop avro read sqoop intro evolution command.

Agenda Day-8

Get and Serialized data reads.

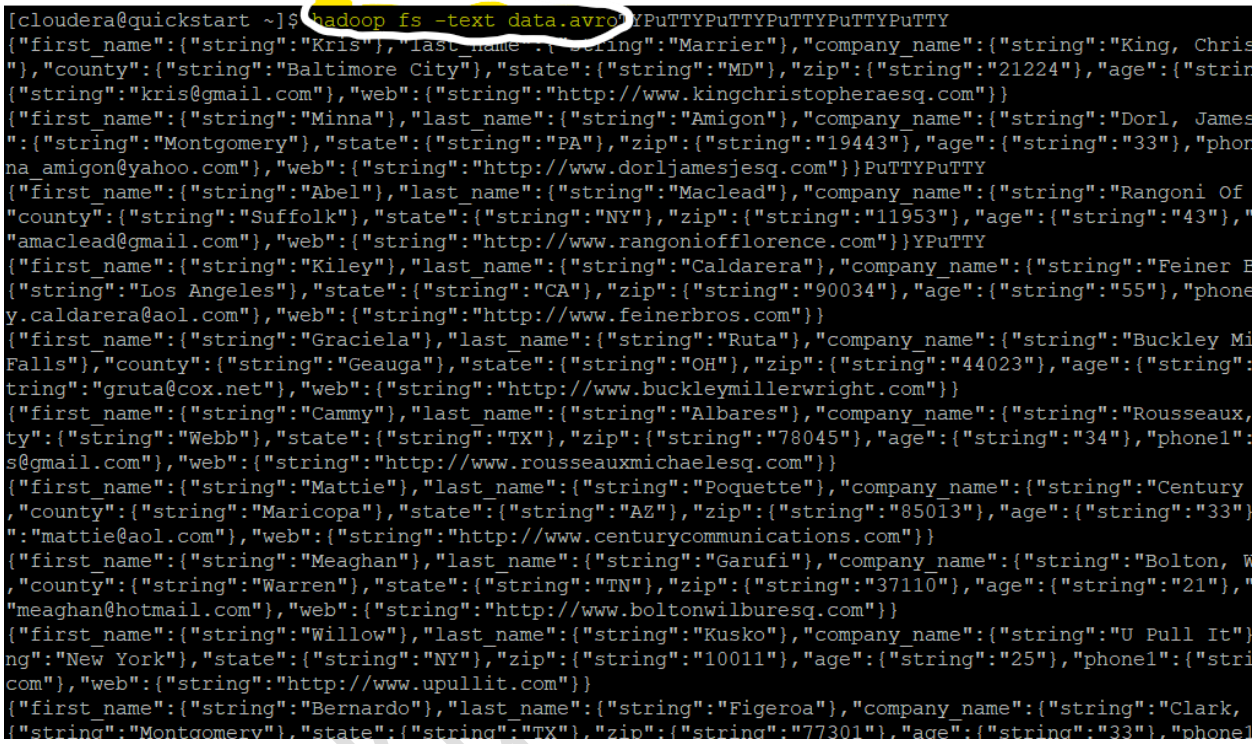
Scenario:

1) Create a directory named “ldir” in /home/cloudera
2) Move a file from hdfs to local directory created. (hdfs fs -get <source> <destination>)



We could be able to read serialized data(.avro) only in HDFS. Below is the command for the same.text is the command available in Hadoop to deserialize the data and read.

hadoop fs -text <path_of the serialized file>



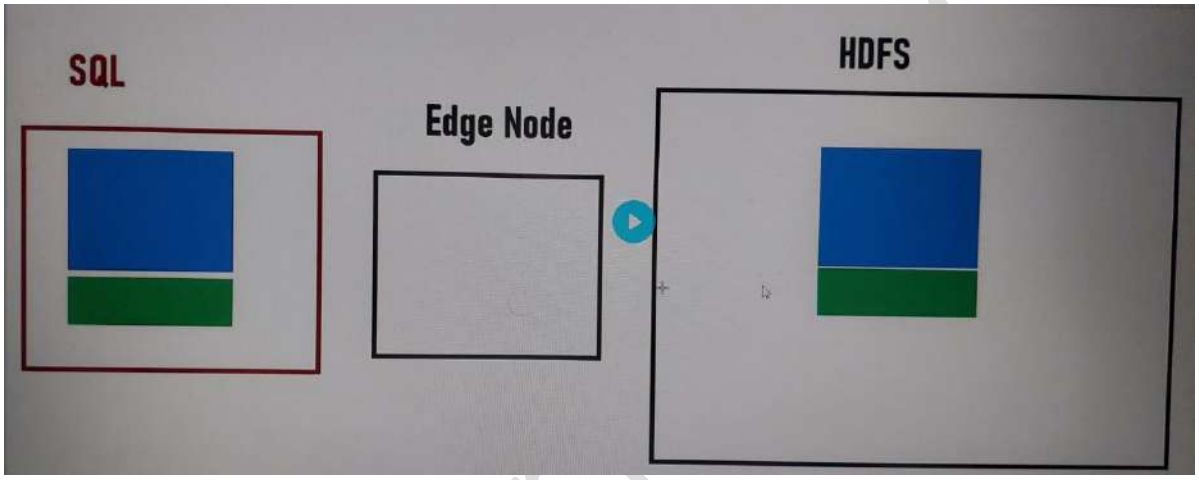
Sqoop Intro Evolution

- Hadoop was introduced in 2006 and By 2018 the biggest challenge is to get the large data dumped in SQl to Hadoop (RDBMS and DW runs on SQL)
- SQL and Hadoop are quite opposite. Sql is single server, hadoop is distributed server,SQL has ETL – hadoop has ELT, SQL Supports only structured data – hadoop supports both structured and semi-structured data.

Problems to Transfer Data from SQL to Hadoop:

- There is no direct transfer between SQL and HDFS, it has to happen through Edge Node. Firstly SQL table data to be converted to file and that to be sent edge node and then edge node to HDFS.
  Data Transfer – Slow (used to takes days)

- Portion Data transfer from SQL to HDFS. No solution at that time.
  eg: when a table has India, UK and US data and we want to transfer only India data from table to HDFS.
- Incremental Data Imports – when data from SQL is transferred to HDFS, and then the SQL table got appended with some data, Transfering the appended data to HDFS



- Modified Data imports – After transferring the data from SQL data to HDFS, SQL table data got updated.
- Serialized Data import
- NO Proper Exports – Exporting the data from HDFS to SQL.

Sqoop Promises

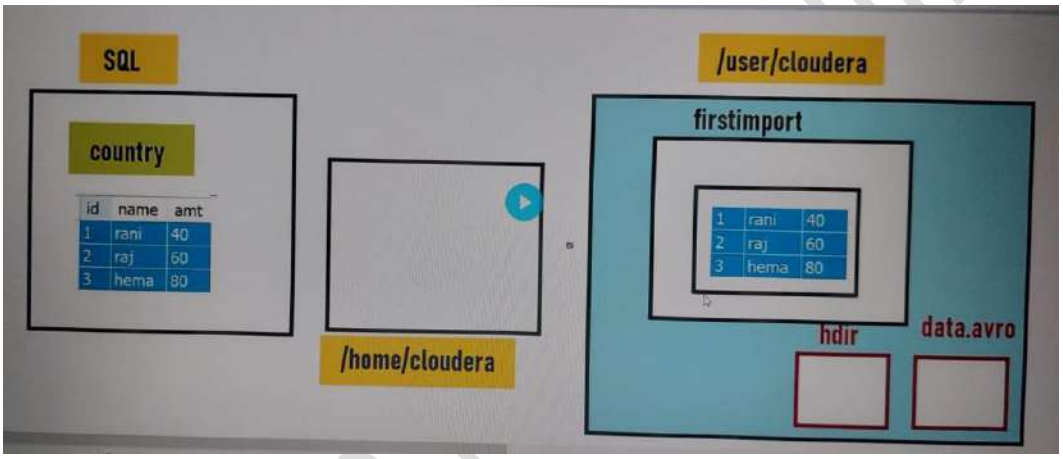
Sqoop was introduced in June 1, 2009 by Kathleen.
She stated that with edge node data will be transferred directly from SQL to HDFS.
And also if checks for updated data in SQL and updates those records in HDFS.

Sqoop is made of Java, Serialization has very good integration with java and Serialized data imports are very much possible.

SQL - -- Hadoop → SQOOP is derived.

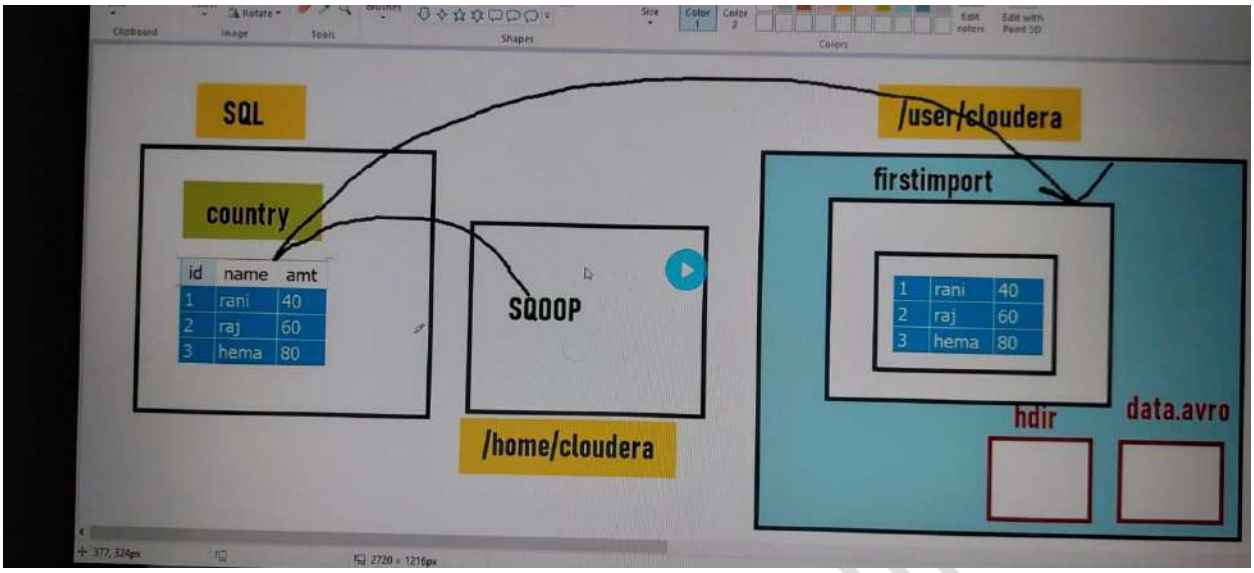
Sqoop Sample Execution

Data from **SQL to** be transferred to Hadoop into a directory firstimport

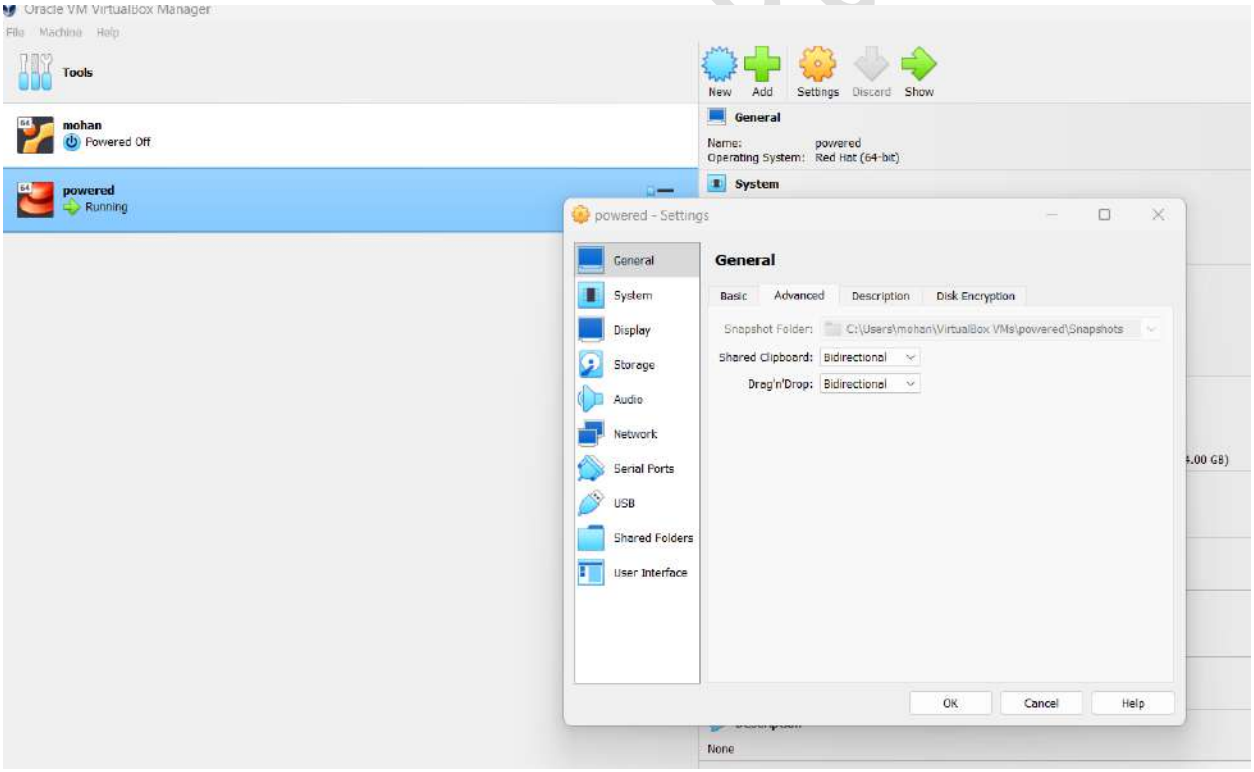


sqoop import --connect jdbc:mysql://zeyodb2.cxxfmhblx5p2.ap-south-1.rds.amazonaws.com/zeyodb –username root –password Aditya908 **–table country** –m 1 –delete-target-dir **–target-dir /user/cloudera/firstimport**
THE ABOVE COMMAND IS NOT WORKING IN CLOUDERA

with the above sqoop in edge node reaches to the table and imports the data to hdfs.



To allow copy and paste in Virtual Box, below is the option



Sqoop Command Design

For Sqoop import – total we need 6 details from SQL and 1 Detail from HDFS

Task 1:
Hadoop dfsadmin -safemode leave
echo zeyobron > /home/cloudera/file1
echo analytics > /home/cloudera/file2
hadoop fs -put /home/cloudera/file1 /user/cloudera
hadoop fs -appendToFile home/cloudera/file2 /user/cloudera/file1
hadoop fs -cat /user/cloudera/file1

Task 2:
What is hadoop federation?

Sqoop Controlling import where
Agenda Day-9

Sqoop Command Design:

- After executing Sqoop Command, when we see **Retrieved <> records**, it means that the job got executed successfully.
- For Sqoop import we need 6 details from SQL as below
  - Ip/hostname
  - Port number
  - Database name
  - Table name
  - Username
  - Password
- For sqoop we need 1detail from which is target Dir

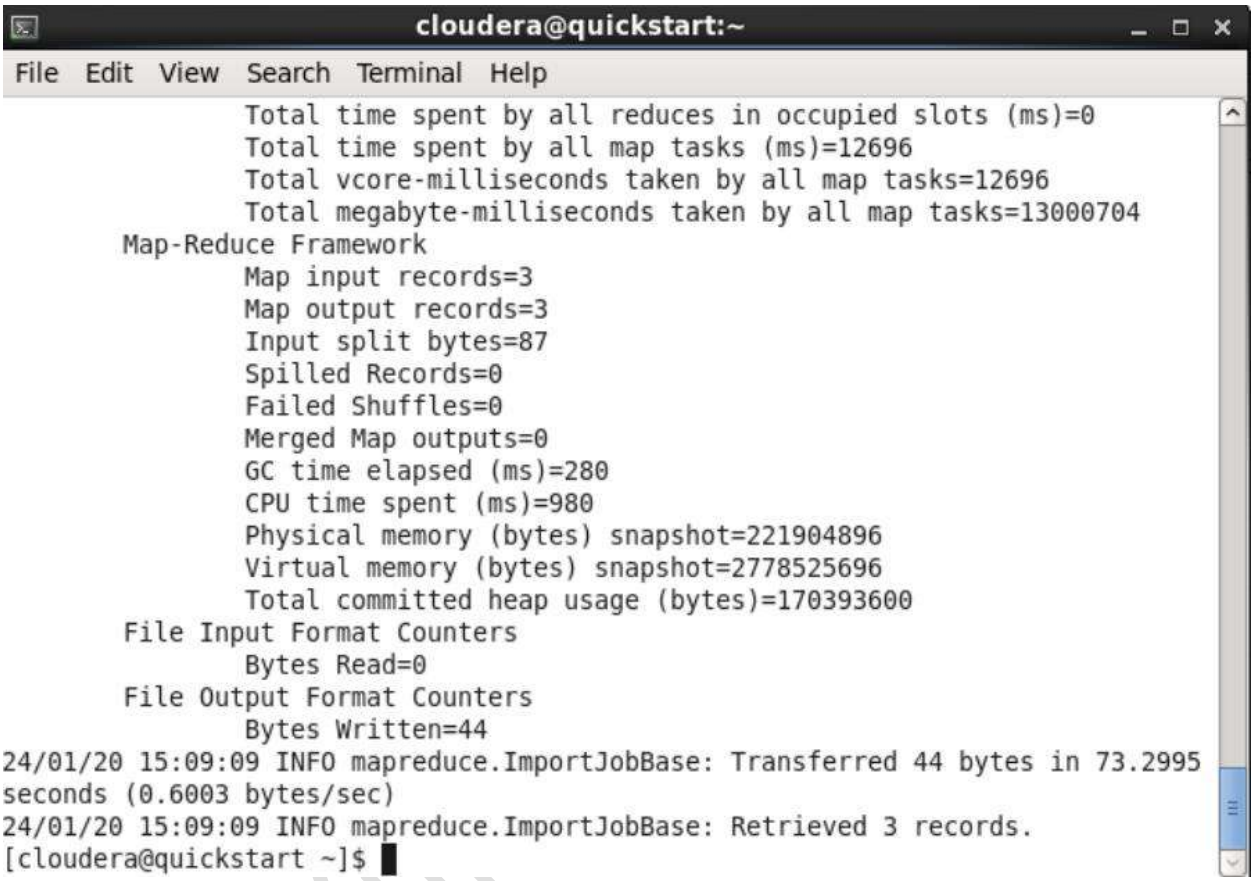
- SQOOP Import template (\ - to execute the command in multiline):

```
sqoop import \
--connect jdbc:mysql://HOSTNAME:PORT/DATABASE \
--username USERNAME \
--password PASSWORD \
--table TABLE \
--m 1 \
--target-dir TARGET
```

LAB Commands:

1. hadoop dfsadmin -safemode leave
2. mysql -uroot -pcloudera
3. create database if not exists zeyodb;
4. use zeyodb;
5. drop table if exists zeyotab;
6. create table zeyotab(id int, name varchar(100), city varchar(100));
7. insert into zeyotab values (1,'sai','chennai');
   insert into zeyotab values (1,'ravi,'chennai');
   insert into zeyotab values (1,'rani,'chennai');
   insert into zeyotab values (1,'vasu,'chennai');
8. select * from zeyotab;
9. quit
10. sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera --m 1 --table zeyotab --delete-target-dir --target-dir /user/cloudera/firstimport
11. hadoop fs -ls /user/cloudera/firstimport

12.hadoop fs -cat /user/cloudera/firstimport/part-m-00000



Sqoop portion imports type1

Using --where, this should be append at the end of the sqoop template.
Eg: --where “city=’chennai’”

Not Explained in this session:
Sqoop will perform this filter on the SQL(RDBMS) side only.

Sqoop portion imports type2
Incremental Intro
Interaction Session

Sqoop Query import incremental imports
Agenda Day-10

Controlling Imports Type-2

Sqoop Promised below Features

- Faster Imports
- Portion Imports
- Incremental Imports
- Modified Imports
- Serialized Data imports
- Exports
- Batch Exports
- Cloud Imports
- Casting Imports

Now under Database, we have two tables, and we want to import this to HDFS and also we should be able to do portion imports.

Importing Using Query:

Use the parameter --query in the SQOOP Template

Template:
Example:
sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera --m 1 --delete-target-dir --target-dir /user/cloudera/queryimport --query "select a.*,b.product from zeyotab a join zeyoprod b on a.id=b.id where \$CONDITIONS and city='city'"

```
mysql> show tables
    -> ;
+------------------+
| Tables_in_zeyodb |
+------------------+
| zeyoprod         |
| zeyotab          |
+------------------+
2 rows in set (0.05 sec)

mysql> select * from zeyoprod
    -> ;
+------+---------+
| id   | product |
+------+---------+
|    1 | mobile  |
|    2 | laptop  |
|    3 | mouse   |
|    4 | water   |
+------+---------+
4 rows in set (0.04 sec)

mysql> select * from zeyotab;
+------+------+-----------+
| id   | name | city      |
+------+------+-----------+
|    1 | sai  | chennai   |
|    2 | ravi | hyderabad |
|    3 | rani | chennai   |
|    4 | vasu | banglore  |
+------+------+-----------+
4 rows in set (0.03 sec)

mysql> select a.*, b.product from zeyotab a join zeyoprod b on a.id=b.id
    -> ;
+------+------+-----------+---------+
| id   | name | city      | product |
+------+------+-----------+---------+
|    1 | sai  | chennai   | mobile  |
|    2 | ravi | hyderabad | laptop  |
|    3 | rani | chennai   | mouse   |
|    4 | vasu | banglore  | water   |
+------+------+-----------+---------+
4 rows in set (0.03 sec)

mysql>
```

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera --m 1 --delete-target-dir --target-dir /user/cloudera/queryimport --qu
ery "select a.*,b.product from zeyotab a join zeyoprod b on a.id=b.id where \$CONDITIONS"
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
24/01/21 10:10:13 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.12.0
24/01/21 10:10:13 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
24/01/21 10:10:14 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
24/01/21 10:10:14 INFO tool.CodeGenTool: Beginning code generation
24/01/21 10:10:14 INFO manager.SqlManager: Executing SQL statement: select a.*,b.product from zeyotab a join zeyoprod b on a.id=b.id where  (1 = 0)
24/01/21 10:10:14 INFO manager.SqlManager: Executing SQL statement: select a.*,b.product from zeyotab a join zeyoprod b on a.id=b.id where  (1 = 0)
24/01/21 10:10:15 INFO manager.SqlManager: Executing SQL statement: select a.*,b.product from zeyotab a join zeyoprod b on a.id=b.id where  (1 = 0)
24/01/21 10:10:15 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
```

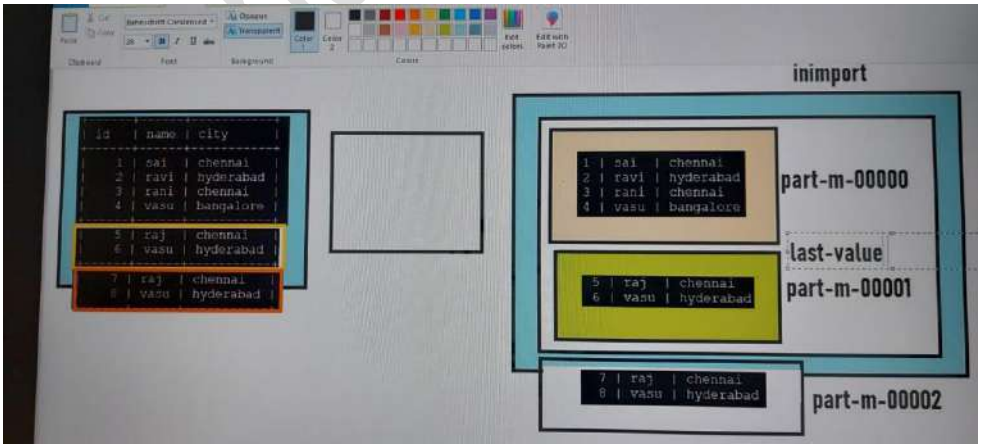
```
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera
Found 4 items
-rw-r--r--   1 cloudera cloudera      29156 2024-01-20 12:18 /user/cloudera/data.avro
drwxr-xr-x   - cloudera cloudera          0 2024-01-20 15:09 /user/cloudera/firstimport
drwxr-xr-x   - cloudera cloudera          0 2024-01-20 09:48 /user/cloudera/mohan
drwxr-xr-x   - cloudera cloudera          0 2024-01-21 10:11 /user/cloudera/queryimport
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/queryimport
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2024-01-21 10:11 /user/cloudera/queryimport/_SUCCESS
-rw-r--r--   1 cloudera cloudera         88 2024-01-21 10:11 /user/cloudera/queryimport/part-m-00000
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/queryimport/part-m-00000
1,sai,chennai,mobile
2,ravi,hyderabad,laptop
3,rani,chennai,mouse
4,vasu,banglore,water
[cloudera@quickstart ~]$
```

Incremental Imports

Incremental import is possible with 3 extra parameters.

- Incremental Type
- Column to consider.
- Last value

sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera --m 1 --table zeyotab --target-dir /user/cloudera/inimport **--incremental append --check-column id --last-value 4**



For incremental to happen we always have to have incremental id or timestamp.

Task 1:
Create if not exists zeyodb
Create zeyotab table (id,name,city)
Insert 4 records
Import only id name columns – requirement

Solution:
sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera -columns id,name --m 1 --table zeyotab --target-dir /user/cloudera/queryimport **--incremental append --check-column id --last-value 4**

Task 2:
What is record container class in sqoop

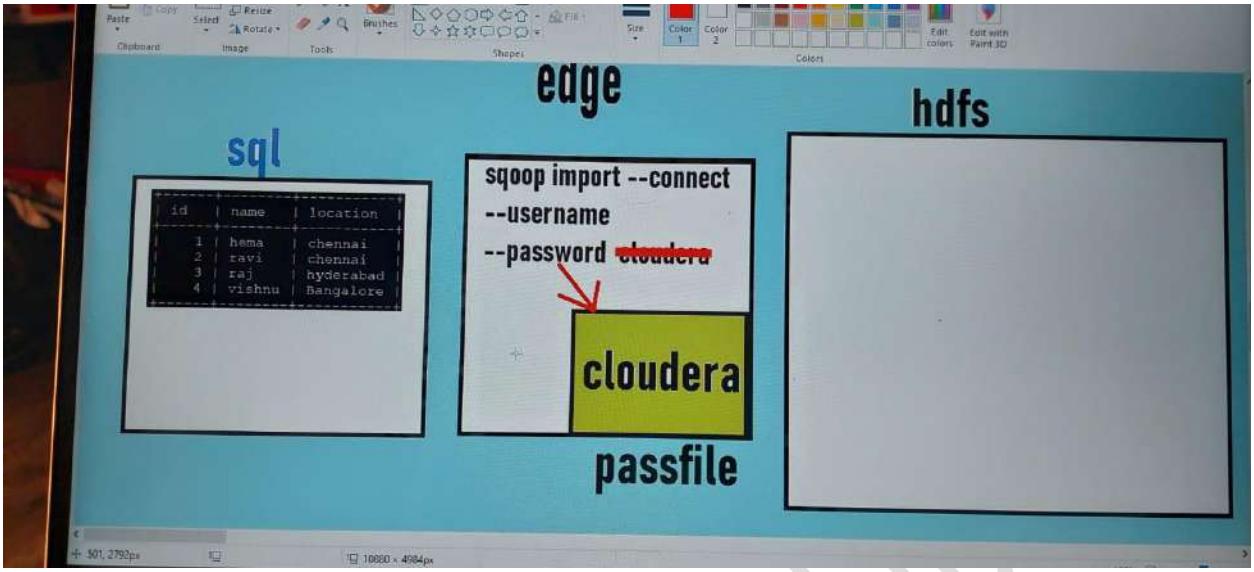
Task 3:
What are the different file formats used in hadoop?

Sqoop Job Automation Multi Mappers Password File Agenda Day 11

Sqoop password file

Passwords should never be hardcoded in the command.

Create a file in Edge Node and keep your password in that file. In the sqoop import command, route the command to read the password from the file created in Edge node. Ensure new line character is not there in the password file.



echo -n cloudera > passfile (-n option to get rid of the new line in the content)

```
[cloudera@quickstart ~]$ cat passfile
cloudera
[cloudera@quickstart ~]$ echo -n cloudera > passfile
[cloudera@quickstart ~]$ cat passfile
cloudera[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
```

Now add password-file argument to sqoop cpmmand.
sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --**password-file file:///home/cloudera/passfile** --m 1 --table zeyotab --delete-target-dir --target-dir /user/cloudera/firstimport

Sqoop Automation Jobs

Scenario:
How will we ensure incremental Imports are happening daily.

We can do this with the help of Sqoop JOB.

Sqoop Job can

- Can take care of the last value.
- Eliminates human intervention.

- Automates the entire SQOOP Import.

Steps:

1) Prepare Sqoop incremental command with last value as 0.
   sqoop import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera -columns id,name --m 1 --table zeyotab --target-dir /user/cloudera/queryimport **--incremental append --check-column id --last-value 0**
2) Create a job named as zeyojob and inside this place the sqoop command. Immediately that job will not down the last value.
   Command to create zeyojob
   sqoop job --create <job_name> -- <import command>
   example: sqoop job --create zeyojob -- import --connect jdbc:mysql://localhost:3306/zeyodb --username root --password cloudera -columns id,name --m 1 --table zeyotab --target-dir /user/cloudera/jdir **--incremental append --check-column id --last-value 0**
3) To see the properties of the job created
   sqoop job --show <job_name>
   Example: sqoop job --show zeyojob

4) Execute job created
   sqoop job --exec <job_name>
   example: sqoop job --exec zeyojob

   once the job is executed, the last value would automatically gets updated and stored.

So whenever we execute the job, the job will be executed and the last-value will automatically gets updated.

```
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
24/01/21 12:56:51 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.12.0
Enter password:
Job: zeyojob
Tool: import
Options:
----------------------------
verbose = false
hcatalog.drop.and.create.table = false
incremental.last.value = 8
db.connect.string = jdbc:mysql://localhost:3306/zeyodb
codegen.output.delimiters.escape = 0
codegen.output.delimiters.enclose.required = false
codegen.input.delimiters.field = 0
```

5) sqoop job --list → to list of the jobs created.

```
[cloudera@quickstart ~]$ sqoop job --list
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
24/01/21 12:54:07 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.12.0
Available jobs:
  zeyojob
[cloudera@quickstart ~]$
```

6) to see where is the last value, below is the command
cat /home/cloudera/.sqoop/metastore.db.script | grep 'last.value'

```
[cloudera@quickstart ~]$ cat /home/cloudera/.sqoop/metastore.db.script | grep 'last.value'
INSERT INTO SQOOP_SESSIONS VALUES('zeyojob','incremental.last.value','8','SqoopOptions')
[cloudera@quickstart ~]$
```

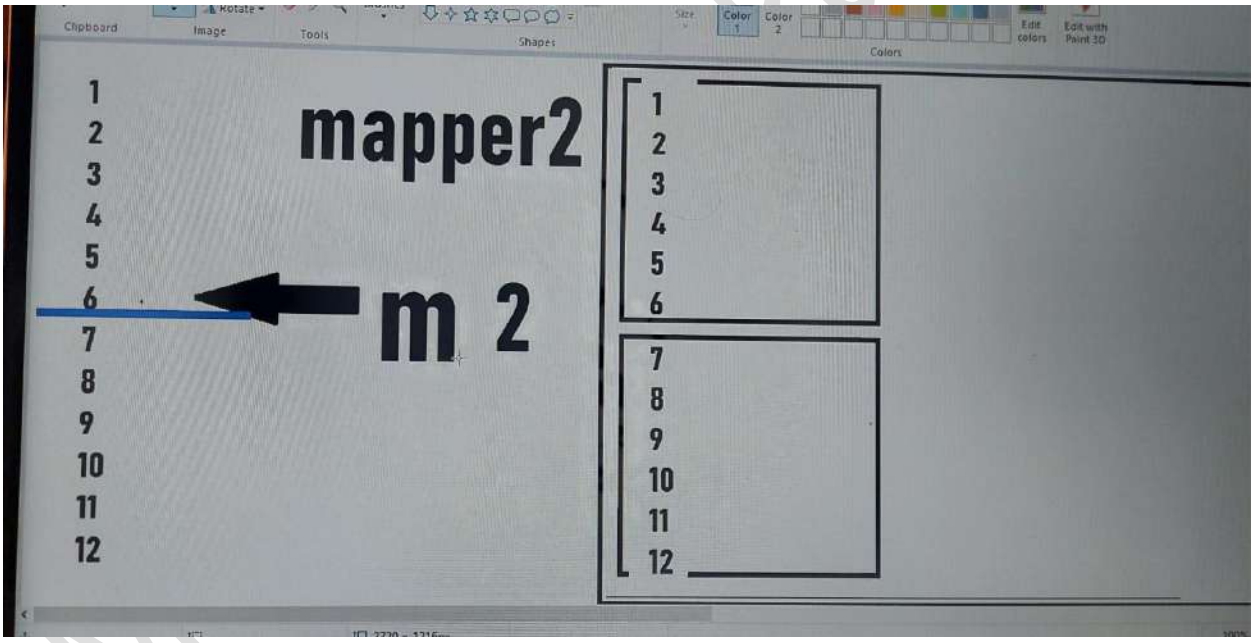
Mappers:

Mappers are nothing based on the number of mappers the tasks will be distributed.

Sqoop Command for the mappers:

sqoop import --connect jdbc:mysql://localhost/zeyodb --username root --password cloudera **--m 2 --split-by id** --table mtab --target-dir /user/cloudera/mapper2

2mappers – 6/6
3mappers – 4/4/4
4mappers – 3/3/3/3



```
                Bytes Written=134
24/01/21 13:24:46 INFO mapreduce.ImportJobBase: Transferred 134 bytes in 53.3214 seconds (2.5131 bytes/sec)
24/01/21 13:24:46 INFO mapreduce.ImportJobBase: Retrieved 8 records.
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/
Found 8 items
drwxr-xr-x   - cloudera cloudera          0 2024-01-21 12:51 /user/cloudera/_sqoop
-rw-r--r--   1 cloudera cloudera      29156 2024-01-20 12:18 /user/cloudera/data.avro
drwxr-xr-x   - cloudera cloudera          0 2024-01-20 15:09 /user/cloudera/firstimport
drwxr-xr-x   - cloudera cloudera          0 2024-01-21 11:06 /user/cloudera/inimport
drwxr-xr-x   - cloudera cloudera          0 2024-01-21 13:24 /user/cloudera/mapper2
drwxr-xr-x   - cloudera cloudera          0 2024-01-20 09:48 /user/cloudera/mohan
drwxr-xr-x   - cloudera cloudera          0 2024-01-21 12:43 /user/cloudera/passwordwithout
drwxr-xr-x   - cloudera cloudera          0 2024-01-21 12:51 /user/cloudera/queryimport
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/mapper2
Found 3 items
-rw-r--r--   1 cloudera cloudera          0 2024-01-21 13:24 /user/cloudera/mapper2/_SUCCESS
-rw-r--r--   1 cloudera cloudera         62 2024-01-21 13:24 /user/cloudera/mapper2/part-m-00000
-rw-r--r--   1 cloudera cloudera         72 2024-01-21 13:24 /user/cloudera/mapper2/part-m-00001
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/mapper2/part-m-00000
-rw-r--r--   1 cloudera cloudera         62 2024-01-21 13:24 /user/cloudera/mapper2/part-m-00000
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/mapper2/part-m-00000
1,sai,chennai
2,ravi,hyderabad
3,rani,chennai
4,vasu,banglore
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/mapper2/part-m-00001
5,mohan,chennai
6,ram,hyderabad
7,prerana,chennai
8,pulibanti,hyderabad
[cloudera@quickstart ~]$
```

Task 1:

Practice mappers above with 2mappers, 4 mappers, no mappers

Task 2: what is default number of mappers in sqoop? – default numbers of mapper 4

Task 3: Maximum Mappers how can I give? – Hypotitical, we can't decide.

it should be decided based on

- Data Size
- Cluster Capacity
- Connections available in SQL
- Do Trail and Error

AWS Azure GCP cloud sqoop imports
Agenda Day 12

Cloud Intro (AWS, Azure, GCP)

Just pay for the time we use it.

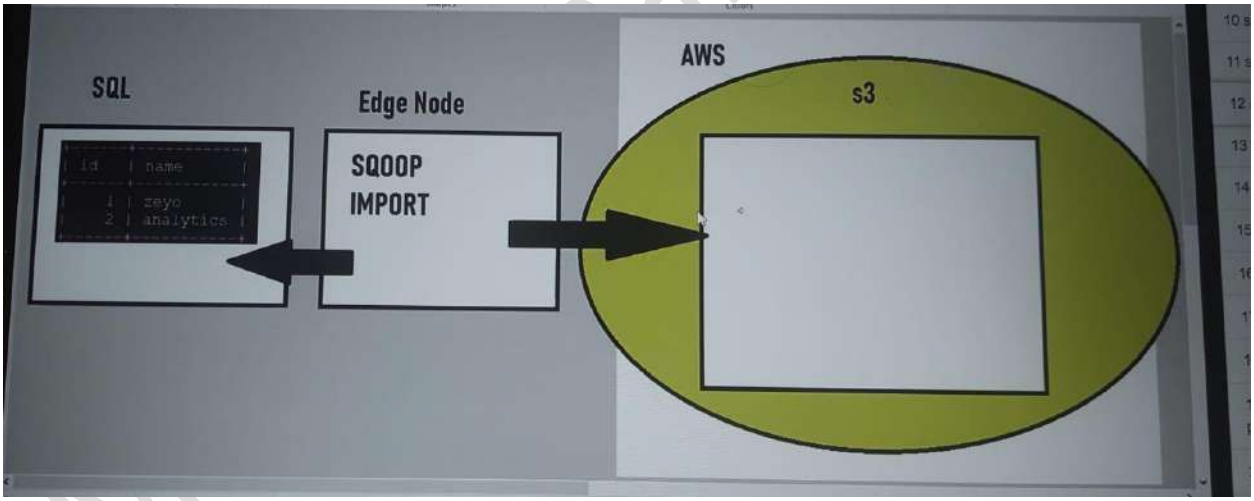
Hadoop Service names in Different Cloud Environments:

AWS → EMR
Azure → HDInsights
GCP → DataProc

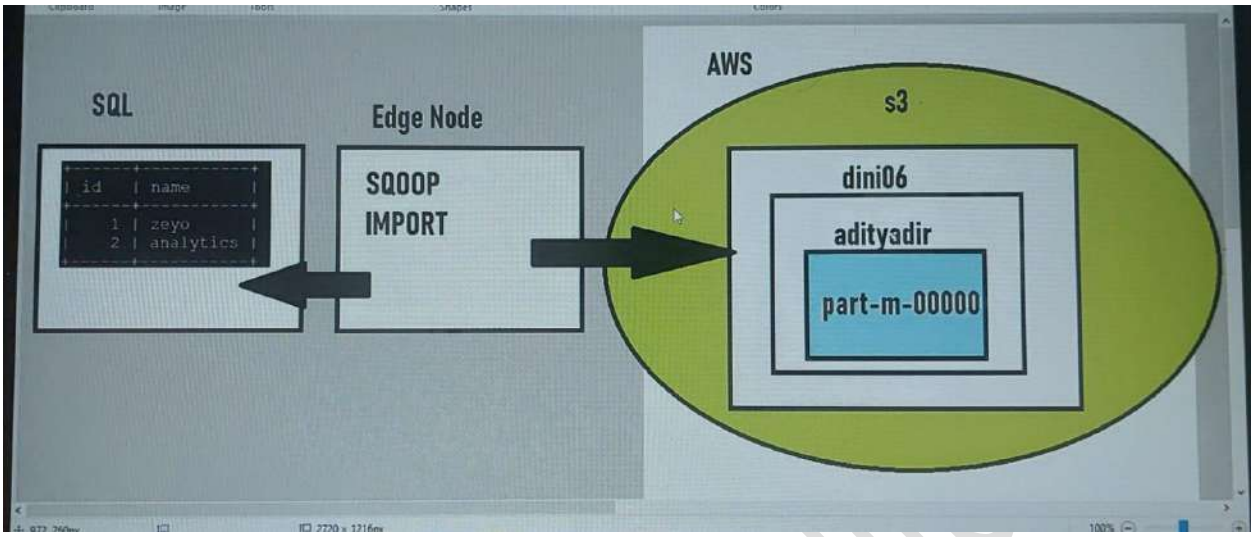
RDS → AWS SQL Service

AWS Cloud imports Handson

Scenario: import Data from sql to Amazon Simple Storage Service(S3)



In S3, data to be stored in a bucket named dini06, and inside bucket a folder named adityadir directory



To upload the data s3 from SQl, we need 3 details and we need to pass to Sqoop template command.

- Access Key
- Secret Key
- End Point

Target Location in S3: s3://dini06/adityadir → s3://<bucket_name>/<directory_name>

Sqoop Command to upload to S3 from SQL. S3a → s3Authentication. For multiline commmads each line to be ended with '\'

```
sqoop import
-Dfs.s3a.access.key=AKIA4LMBSU6NLAJWOGHH
-Dfs.s3a.secret.key=1/fNhN6OftzS/0LiARNe u7JLmClSQUirs9WbZ
-Dfs.s3a.endpoint=s3.ap-south-1.amazonaws.com
--connect jdbc:mysql://localhost/zdb
--username root
--password cloudera
--table cust
--m 1
--target-dir  s3a://dini06/adityadir
```

sqoop import -Dfs.s3a.access.key=AKIA4LMBSU6NLAJWOGHH -Dfs.s3a.secret.key=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 -Dfs.s3a.endpoint=s3.ap-south-1.amazonaws.com --connect jdbc:mysql://localhost/zeyodb --username root --password cloudera --table zeyotab --m 1 --target-dir s3a://dini06/kanthadir

Authentication Names in different cloud environments:
AWS – Key Authentication
Azure – Connection String Authentication
GCP – Service Account Authentication

Sqoop Imports parquet(is a specific file format):

sqoop import --connect jdbc:mysql://localhost/zeyodb --username root --password cloudera --table zeyotab --m 1 --delete-target-dir --target-dir /user/cloudera/pardir --as-parquetfile

Serialization Sample Execution File Formats

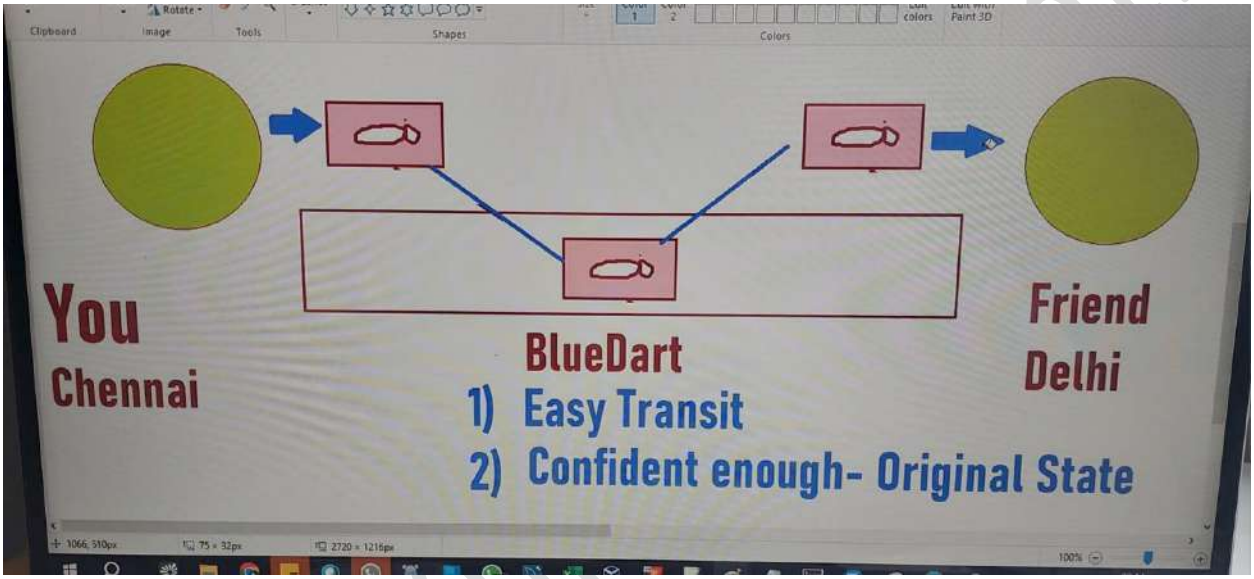
Agenda Day 13

Launch Years of cloud platforms
AWS – 2006
GCP – 2008
Azure – 2010

Serialization Intro

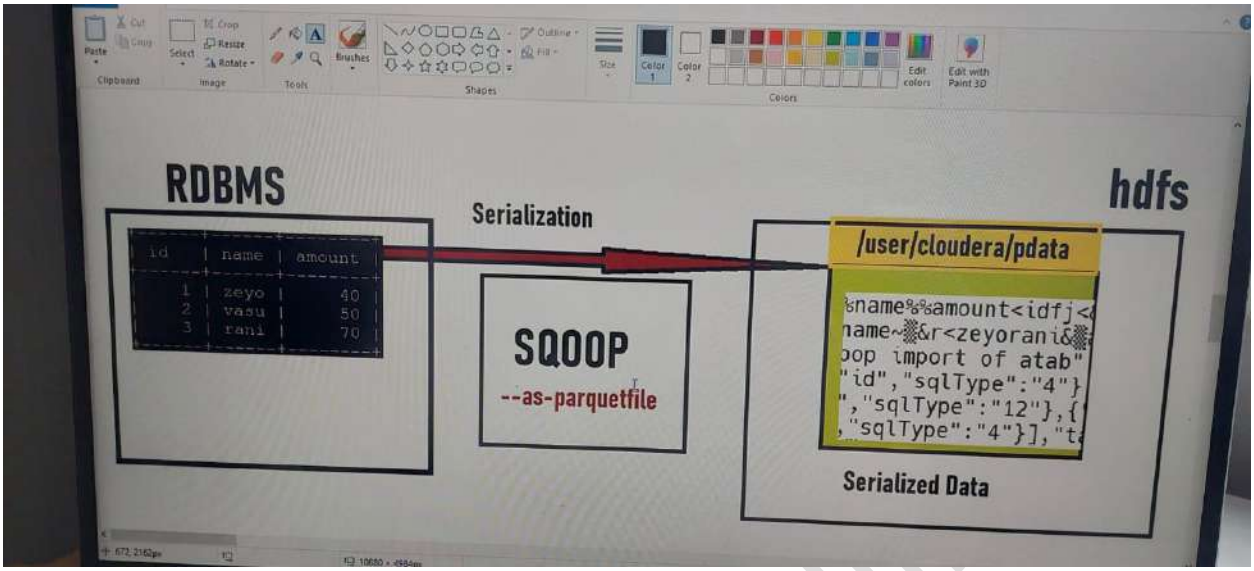
Serialization – changing the nature of raw data for easy transit.
Serialization is used by many tools – Sqoop, Hive , Spark, Kafka,nifi, Cloud, Python, Confluence

Example: Sending air filled balloon from Chennai to Delhi. Removed ballon air for easy transit and we are confident enough that we can bring the balloon back to original state.



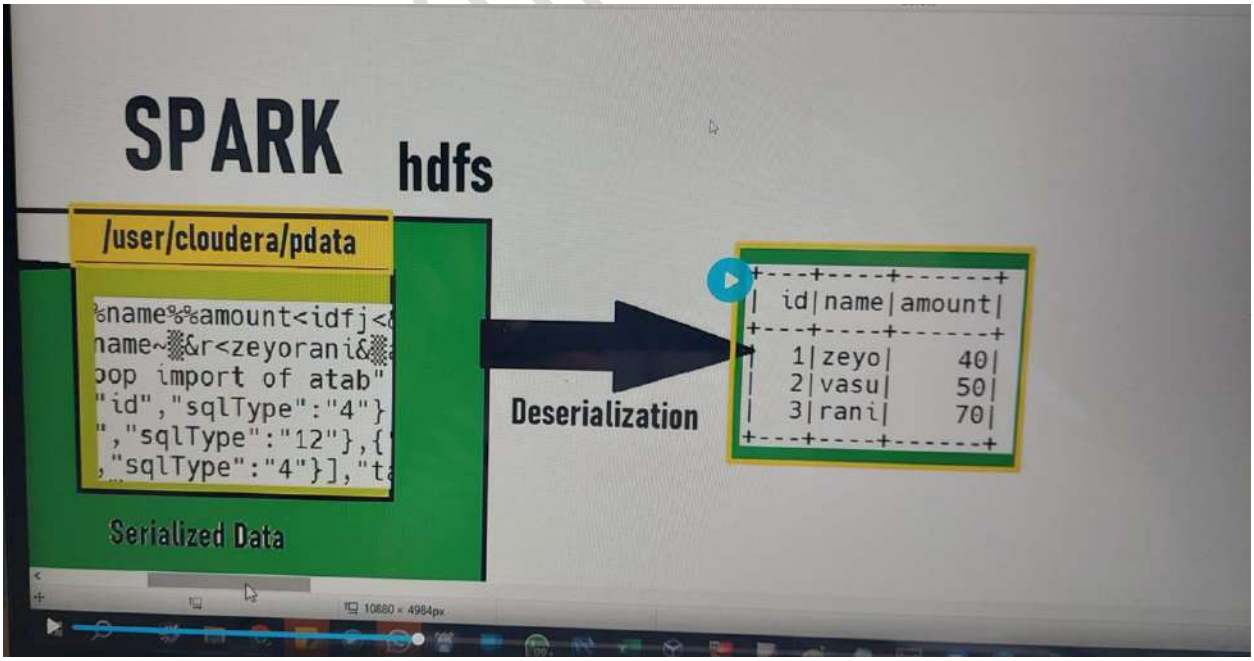
Serialization – Changing raw data to Binary Format
Deserialization – Getting raw data from Binary Formatted/ serialized Data

Serialization Sample Execution



--as-parquetfile converts the Data to serialized and send to HDFS

Spark is one of the tool that will deserialize the data when we execute below command in spark shell.
spark.read.parquet("/user/cloudera/pdata").show()



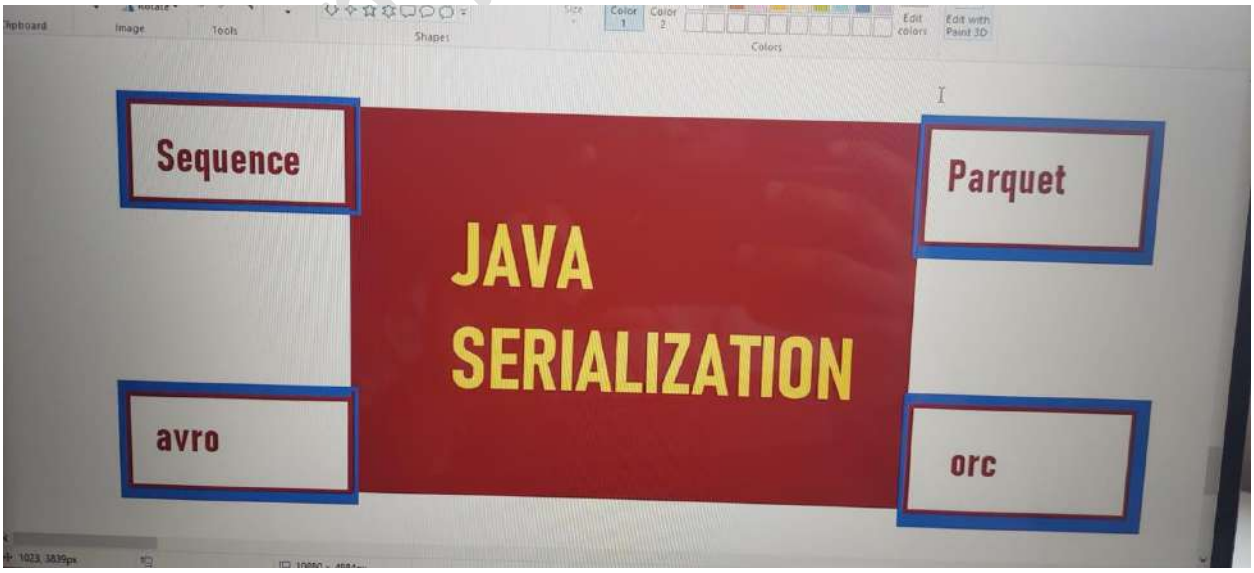
Serialized Advantages

- Easy Transit – Why we need Transit in Big Data?
  for Instance, In Map Reduce – data keeps Shuffling between nodes as we are distributing data and process between different nodes.
- Damn Best Compression
- Faster Querying (if you query something on serialized data outcomes would be much faster)
- Best Storage Mechanism – Data Storage is purely different with serialized data when compared to Raw Data Storage.
- Schema Evolution
- Lighter Data Transfer (we don't require lot of big infrastructure)
- Soul of Big Data

Serialization is found by Sun systems which became soul of big data today.

Serialized File Formats

Serialization started with Java Serialization. Using Java Serialization as base, there are so many file formats being derived.



Every File format has different purpose. Doug Cutting invented Sequence, Parquet, Avro file. LinkedIn introduced orc.

Sequence , RC file formats are no more been used.

Parquet is the king of all the file formats.

Text File:

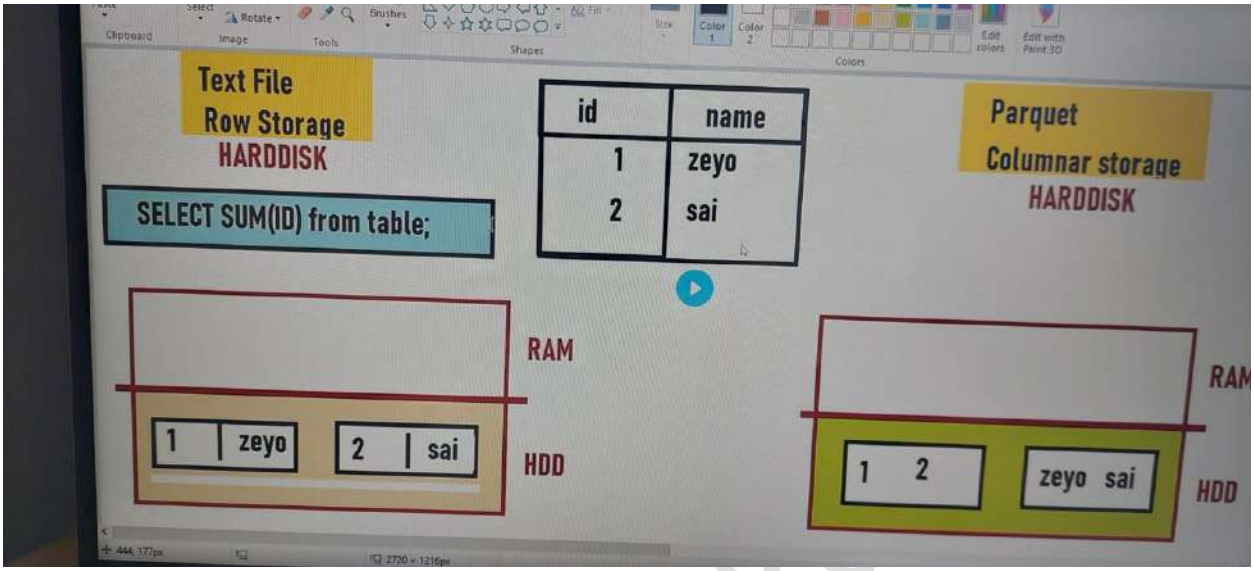
- Readable
- Huge in Size
- Row Storage Mechanism
- Query takes long time.--> Long time to process/Query Text data for Spark and Hive and other tools as it has row storage
- Never used in Big Data World

Parquet:

- Columnar Storage mechanism (file format)
- Because of Columnar Storage, less time to process/Query. Analytics happen only on parque.
- Target Systems → From multiple sources we get the data, but it is always good to store the data in parquet file format as querying would be faster and analyzing
- 60-80% compression ratio.
  Execution:
  Textdata (size 45kb) → parque file (size 5.5mb) To Be Analyzed with Spark

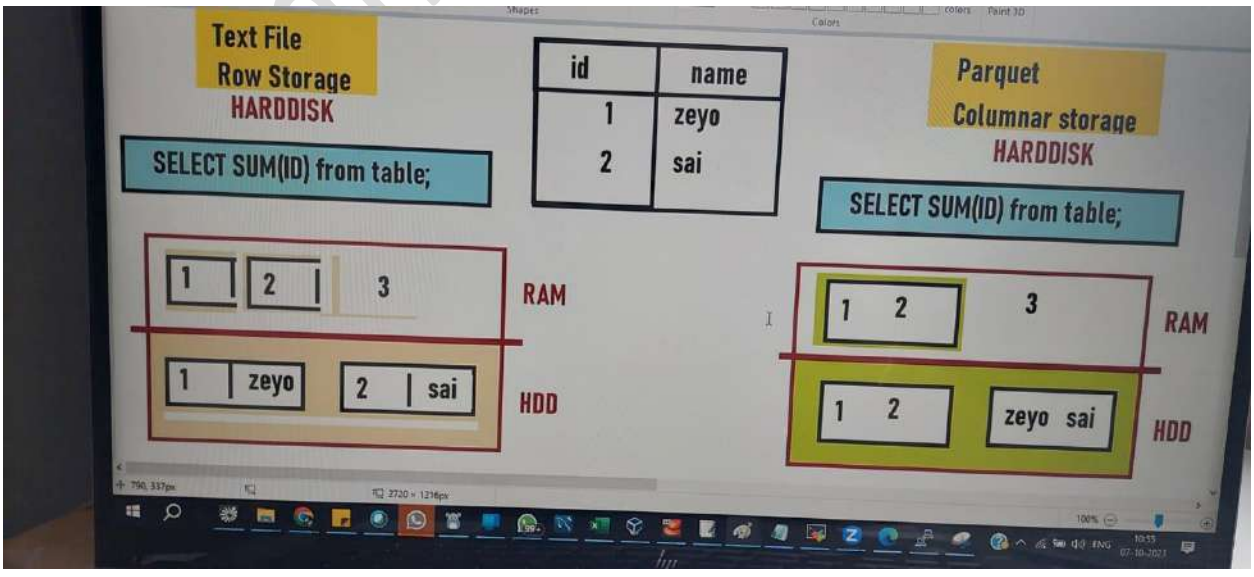
- Predicate Pushdown

Difference between Row Storage and Columnar Storage:



When we suite a query, it brings the data to RAM and process, on the left side of row storage, it has to bring all the data and then should sumup, where as on the right side, we just need to bring the particular column Data.

As all the transformation will be done based on the column, Parquet file Querying will be faster.



ORC (rank 2):

- Columnar Storage Mechanism
- Query is Faster
- 90-95% Compression
- Mostly used for Historical data Storage

But Still Parque is King? → In terms of Querying Parque is faster, Because of ORC compression (ZLIB) where as Parque uses Snappy Compression
Because of high compression rate of ORC, it hinder the Querying Speed.

Tasks:

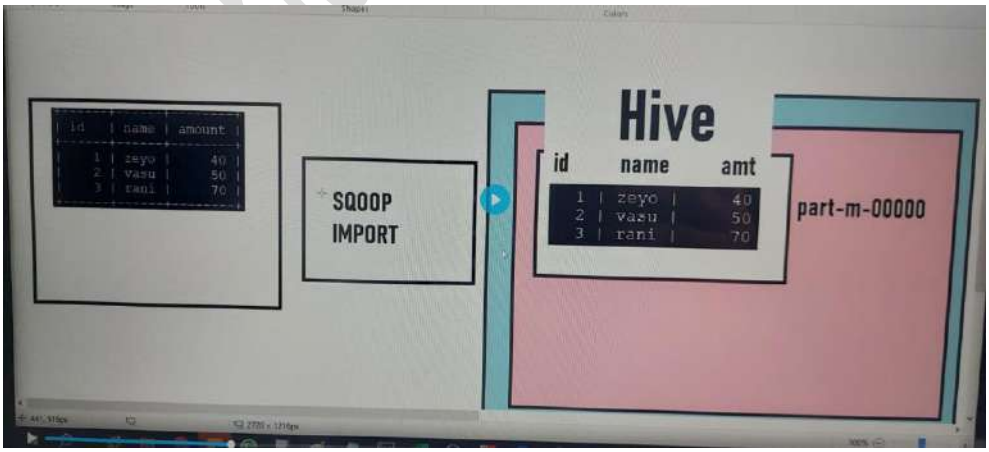
What is Predicate Pushdown?
What is Speculative Execution in Hadoop?
What are the advantages of columnar file format?

Sqoop avro imports export staging hive intro

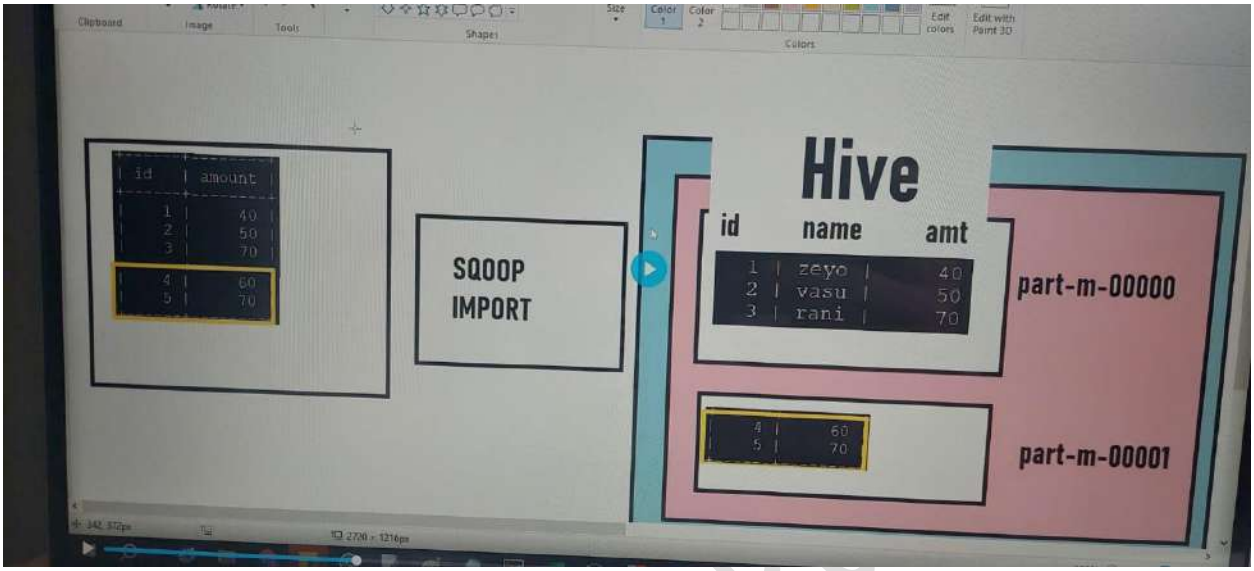
Agenda Day 14

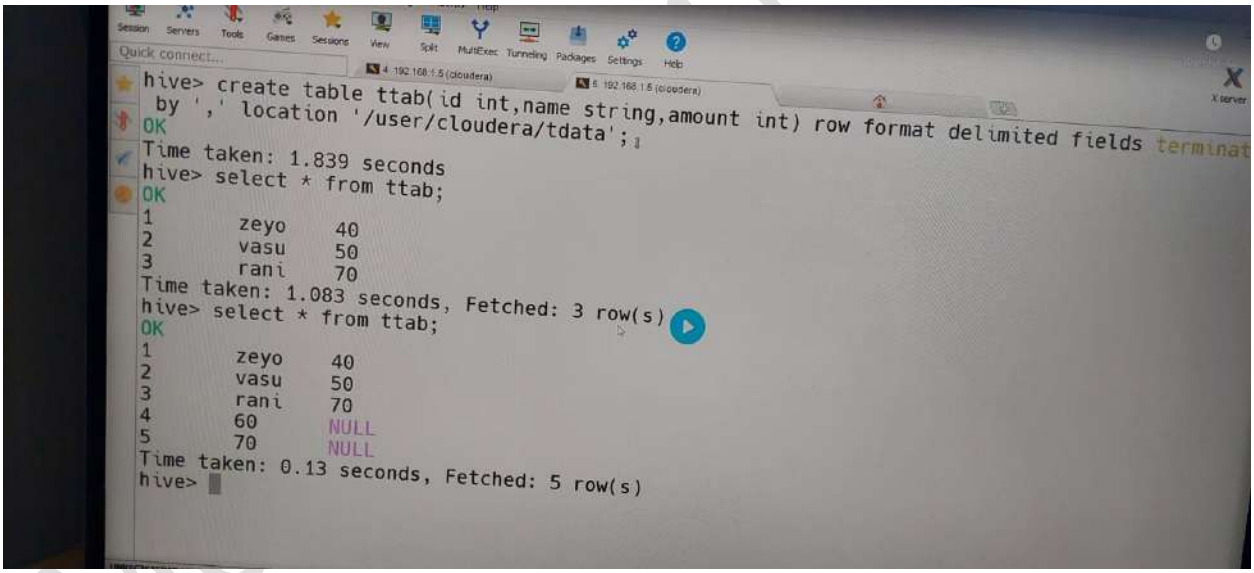
AVRO Schema Evolution

Execution:
Hive is the tool used to process the data and it has the access to create table on top of directory.



Issue: RDBMS column dropped and data increased in RDBMS And the incremental data got imported to HDFS. Schema changed In RDBMS so when Hive process this data it won't be as intended.

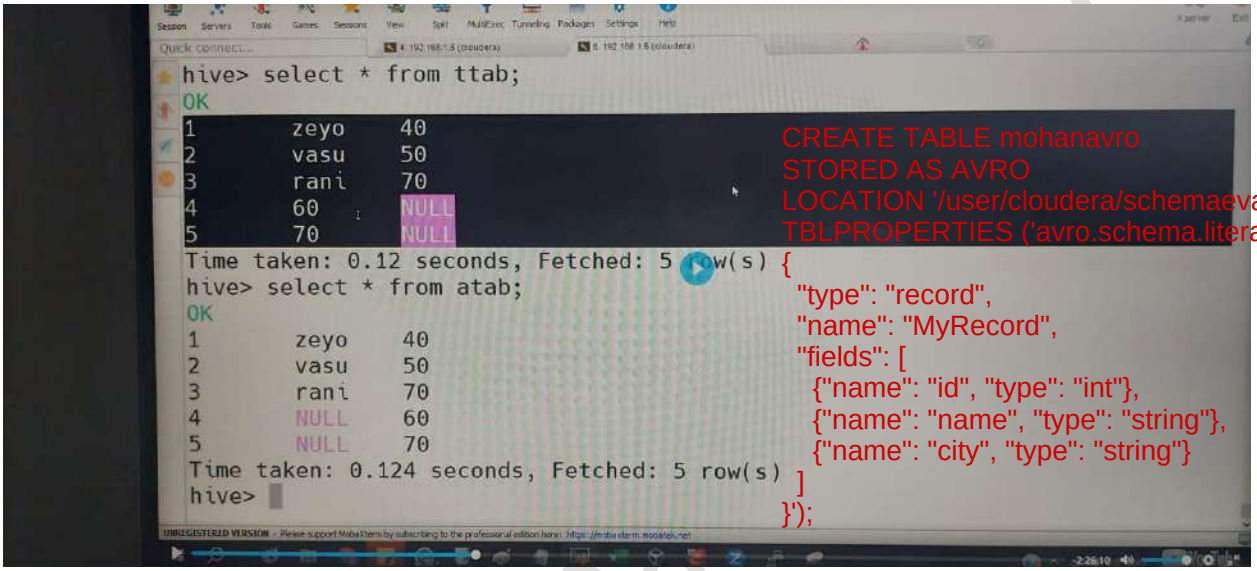




With any Schema changes/evolutions happened on source side, Hive can't withstand those changes.

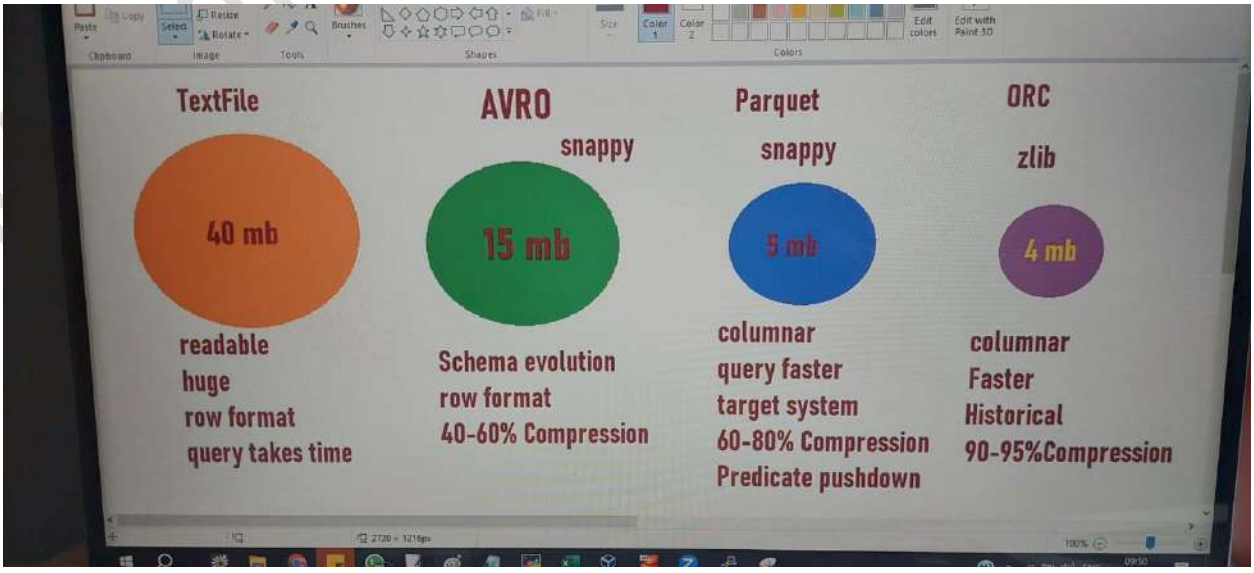
To solve the above problem, AVRO is introduced.

Sqoop cmd to import data in AVRO Format:
sqoop import --connect jdbc:mysql://localhost/ad1 --username root --password cloudera --table atab --m 1 --delete-target-dir --target-dir /user/cloudera/adata --as-avrodatafile

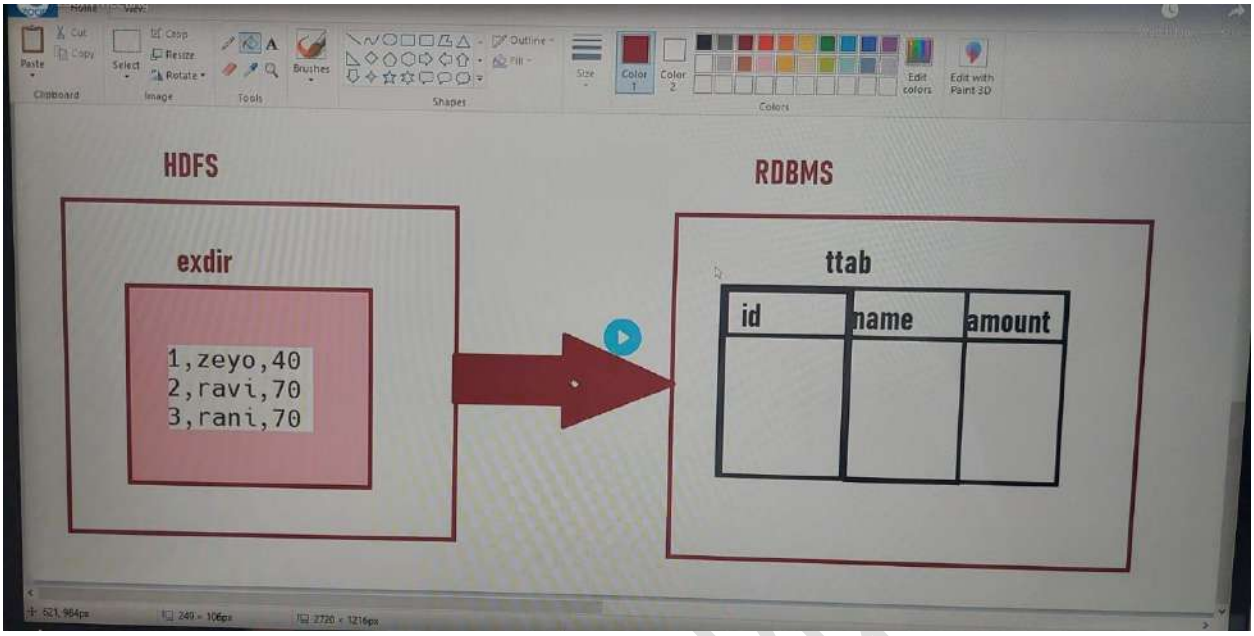


AVRO:

- Schema Evolution
- Row based
- 40-60% compression



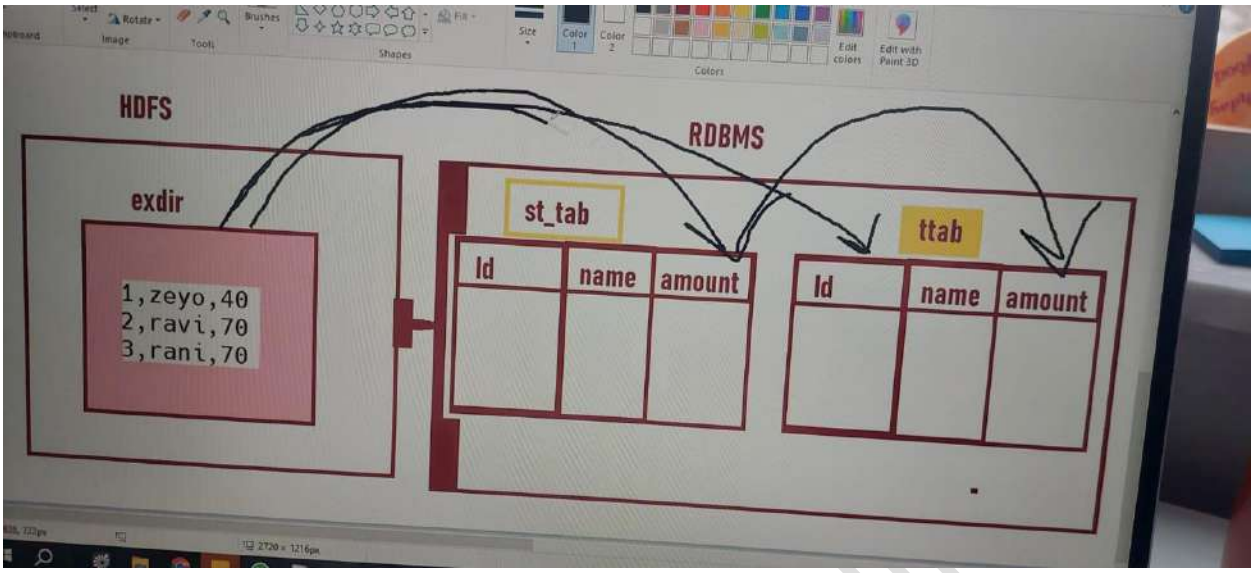
Sqoop Exports and Staging

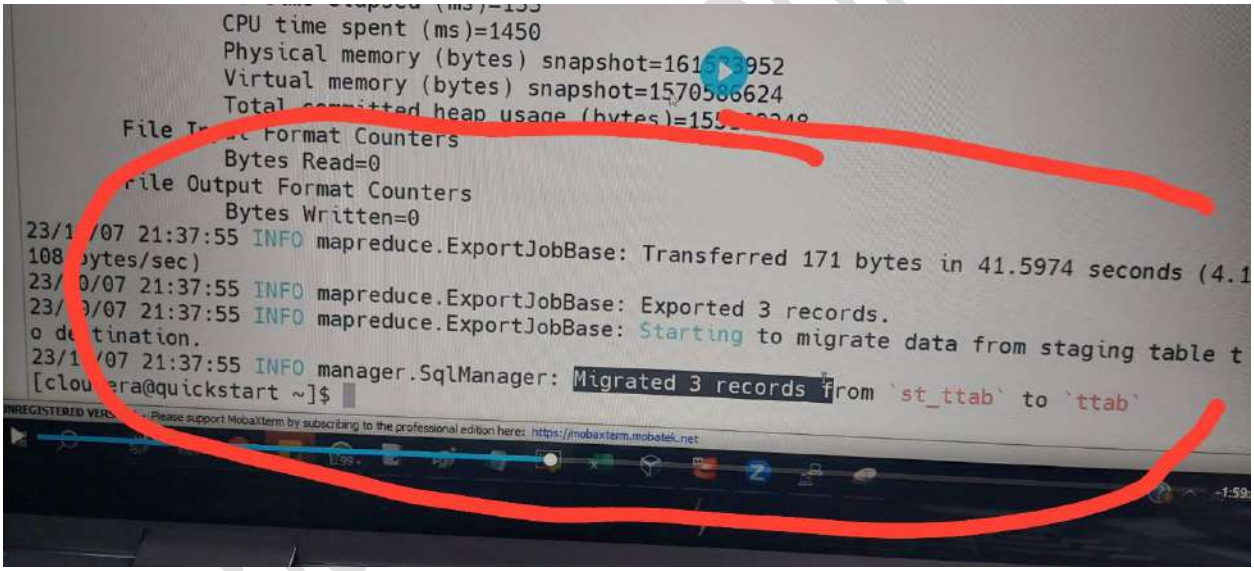


Sqoop Command for Export:
sqoop export --connect jdbc:mysql://localhost/exp --username root --password cloudera --table ttab --m 1 --export-dir /user/cloudera/exdir

In real time, we wil not directly export data to the target tables as we will get partial data exports may happen if there are any network issues or any such issues. If any partial data exports happens to table, its really tough to remove that data. So instead of directly exporting to target, we export to staging first and then

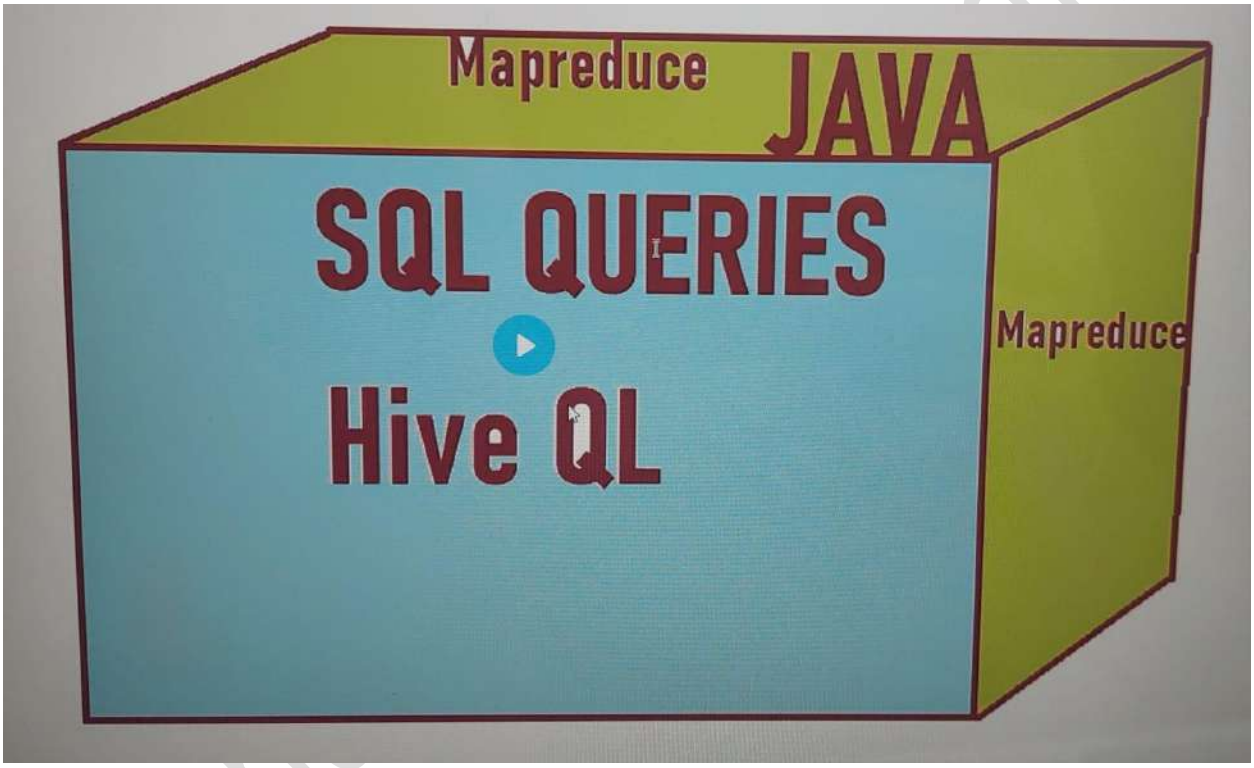
we put to target table.





Hive Intro

2006 - Hadoop Launched (HDFS and Mapreduce)
2007 – MapReduce is made of java. It's very hard to understand.Tough to Handle. Even for small filter you have to write java code.
2008 – Sen Sarma introduced a solution which don't require JAVA. Found a tool that can shoot SQL Queries on HDFS. His tools look like SQL but it's not. You can shoot any sql Queries to process data in HDFS, at the backend the map reduce will run. Every processing engine is made simple, and the tool is HIVE.

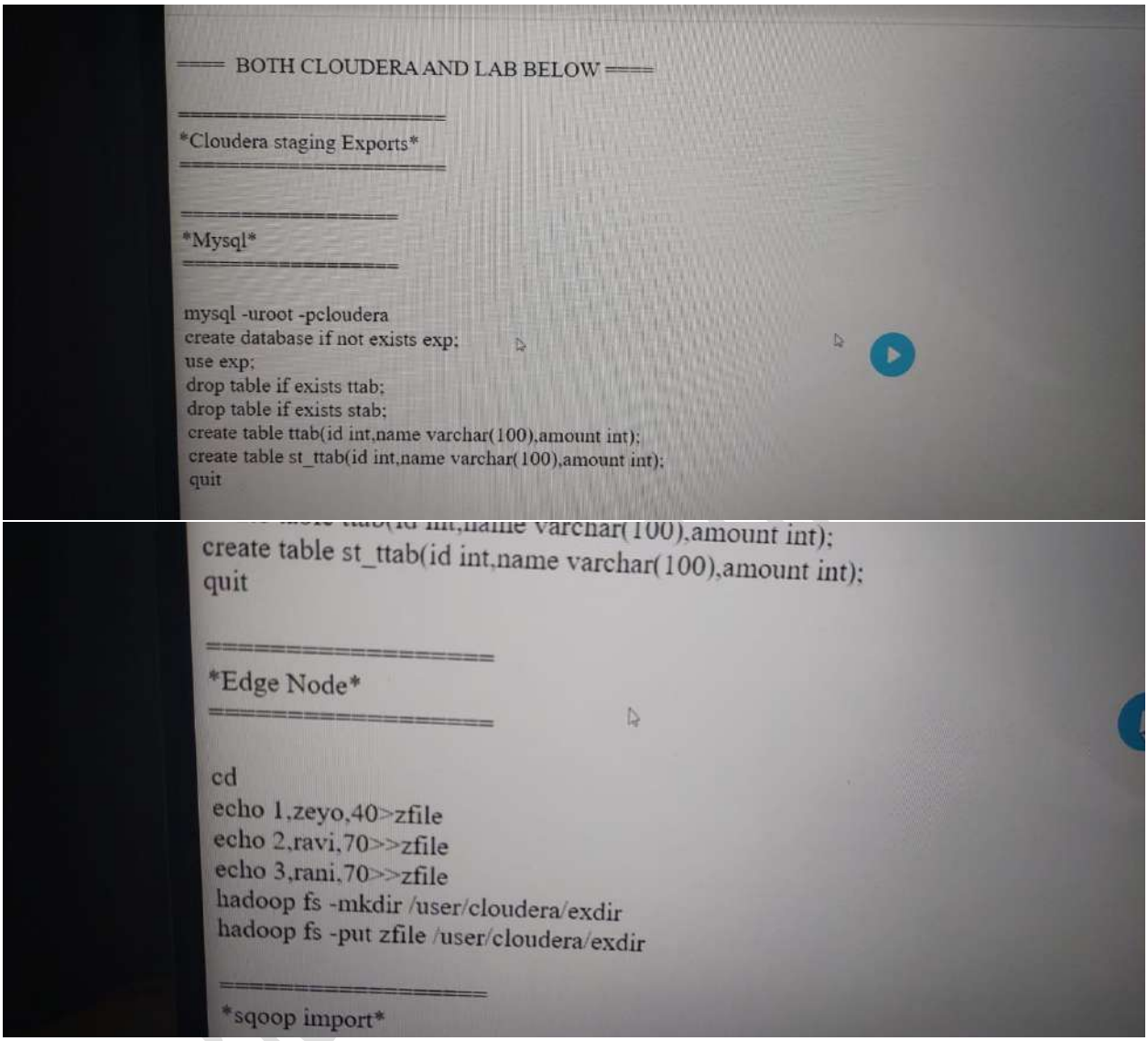


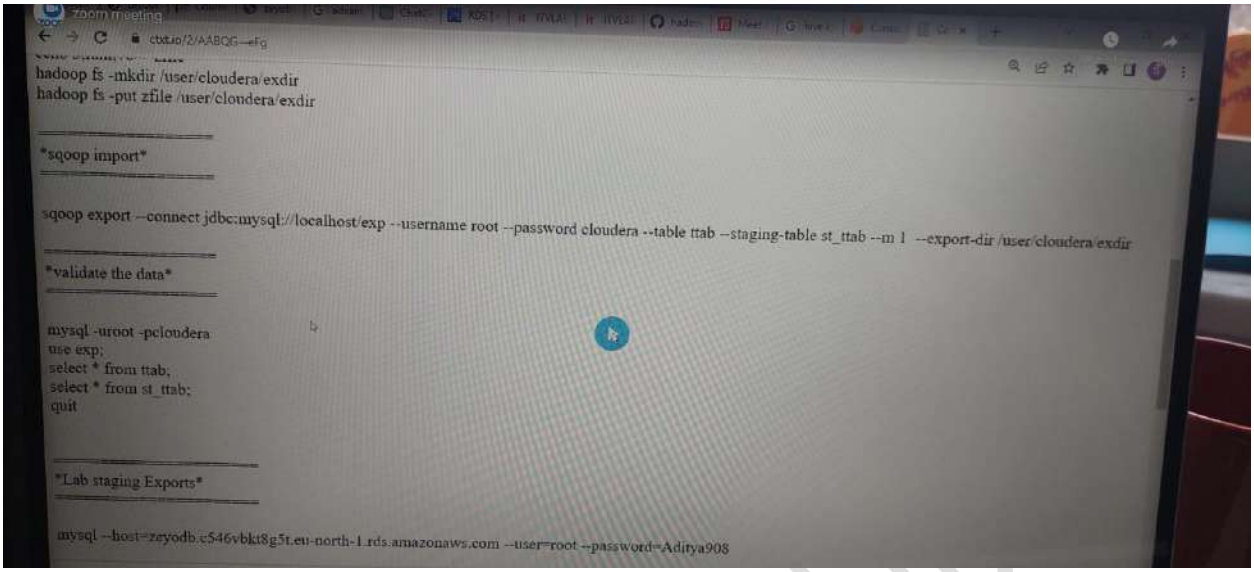
Hive Minor Execution(SQL vs HQL)

Commands of SQL and HQL are same. Only thing is Hive inturn triggers map reduce jobs in the backend.

Using the concept of hive – Redshift, Snowflake, AWS Athena, Synapse has got developed.

Staging Exports Task:





Task 2:
What is Schema on read and schema on write?

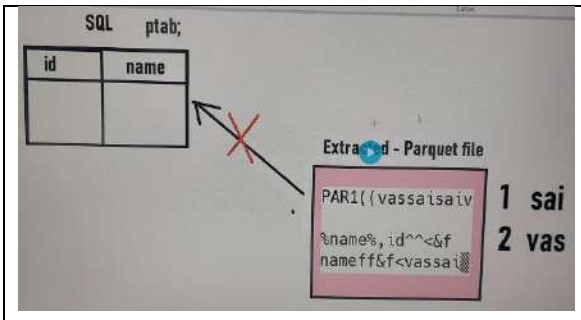
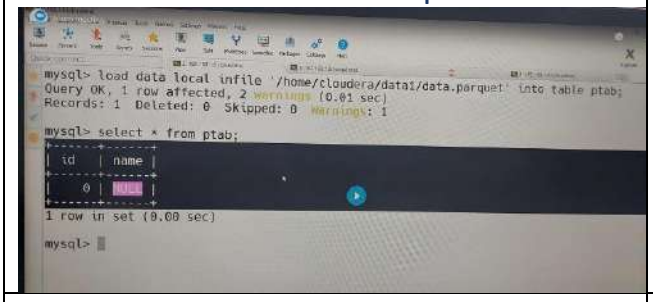
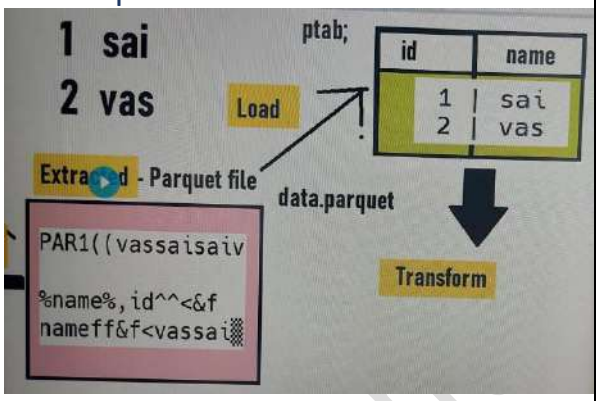
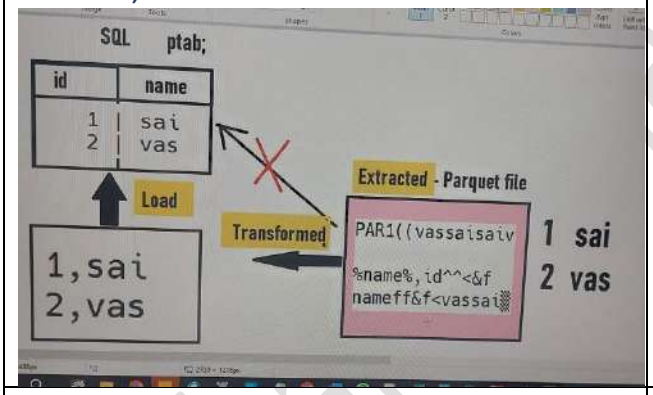
Sqoop Modified Video:
https://www.youtube.com/watch?v=BuhLsf0eXpc

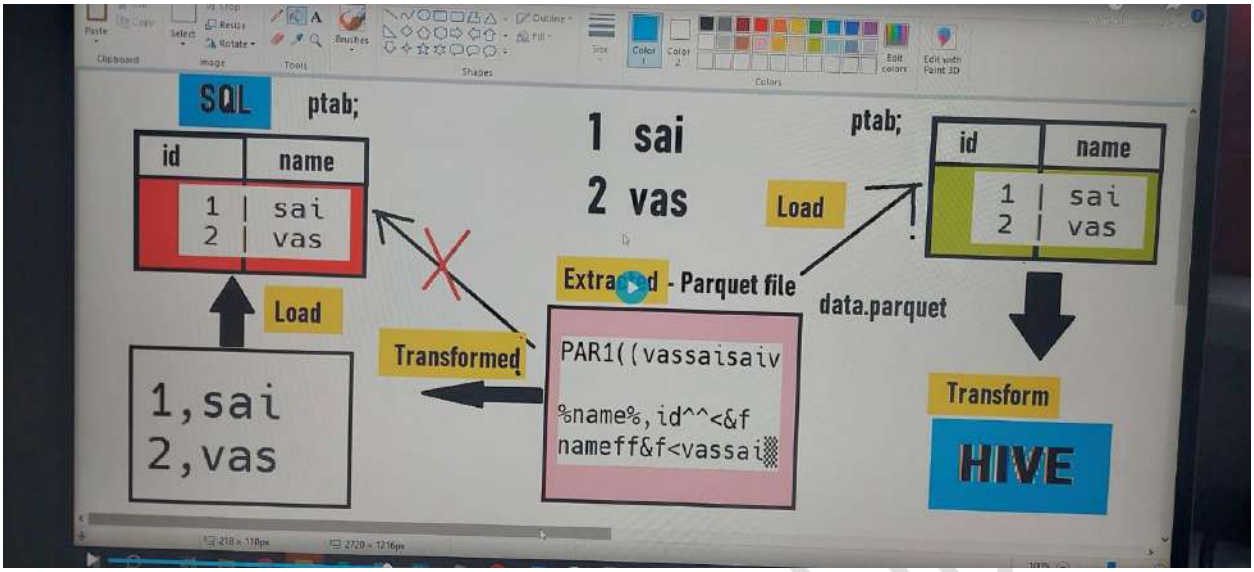
Hive vs SQL deep hive db and tables

Agenda Day 15

SQL vs HQL

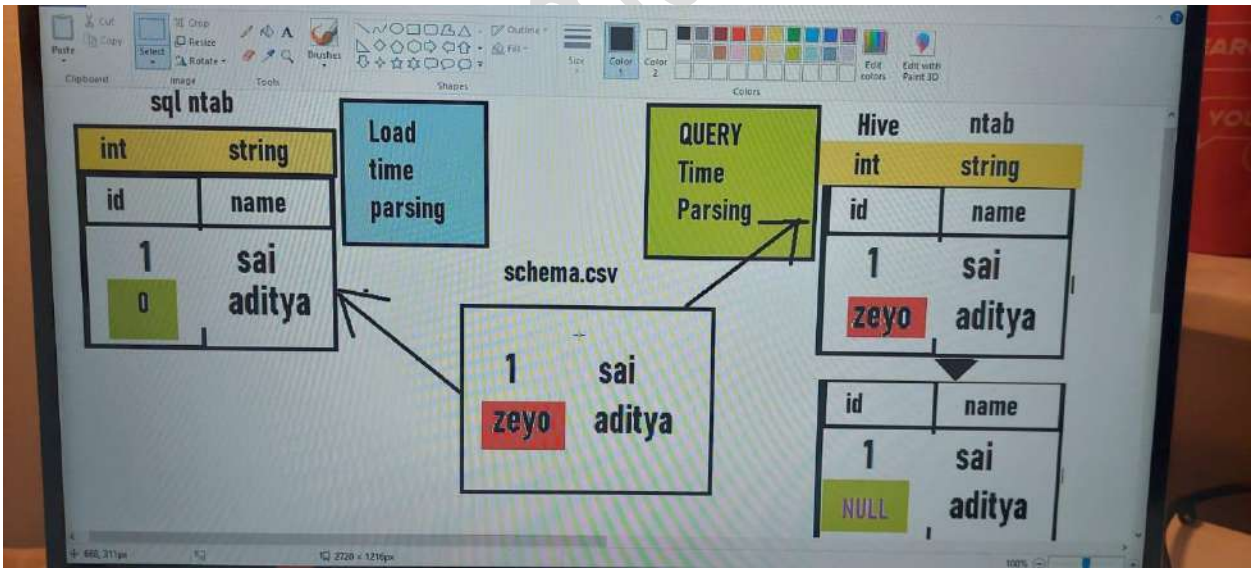
SQL	HQL
Database	Datawarehouse on HDFS. Hive Creates a table on top of a folder and provides the feasibility to query.
Used for ETL (Extract, Transfer, LOAD) purpose	Hive doesn't requires us to transform, Extract, Load and Tranform if required

 When we try to load parquet file to SQL table below will be the output 	ELT Purpose 
Extract , Transform and Load the data 	
Load Time Parsing	Query Time Parsing
Structured	Structured(CSV), Semi-Structured(XML,JSON), Serialized data
Not Good for large data	Not good for Small Data

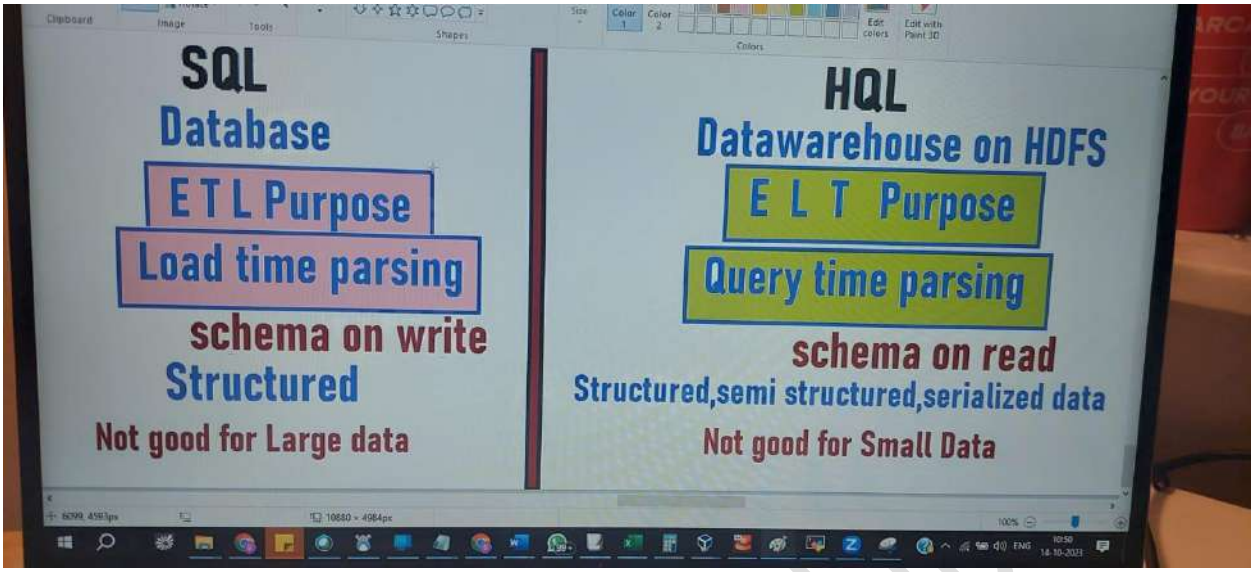


Load Time Parcing (SQL) vs Query Time Parsing (HQL)

Command to load data:
load data local infile '/home/cloudera/data1/schema.csv' into table ntab fields terminated by ',';

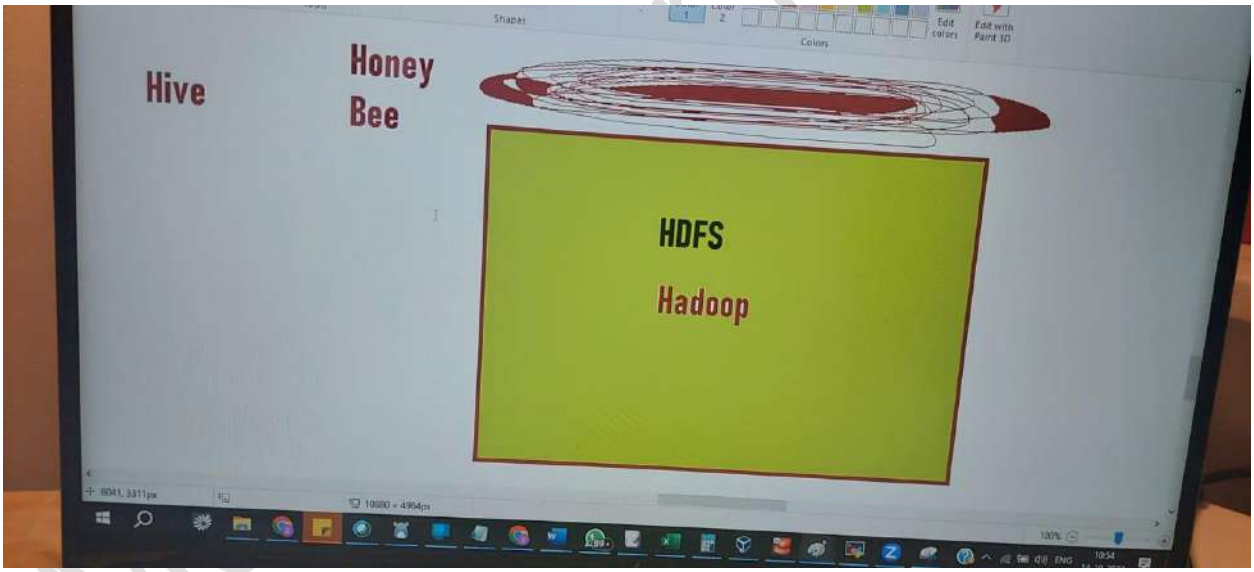


Schema on Write (Load Time Parsing)
Schema on Read (Query Time Parsing)



Hive impacts on HDFS

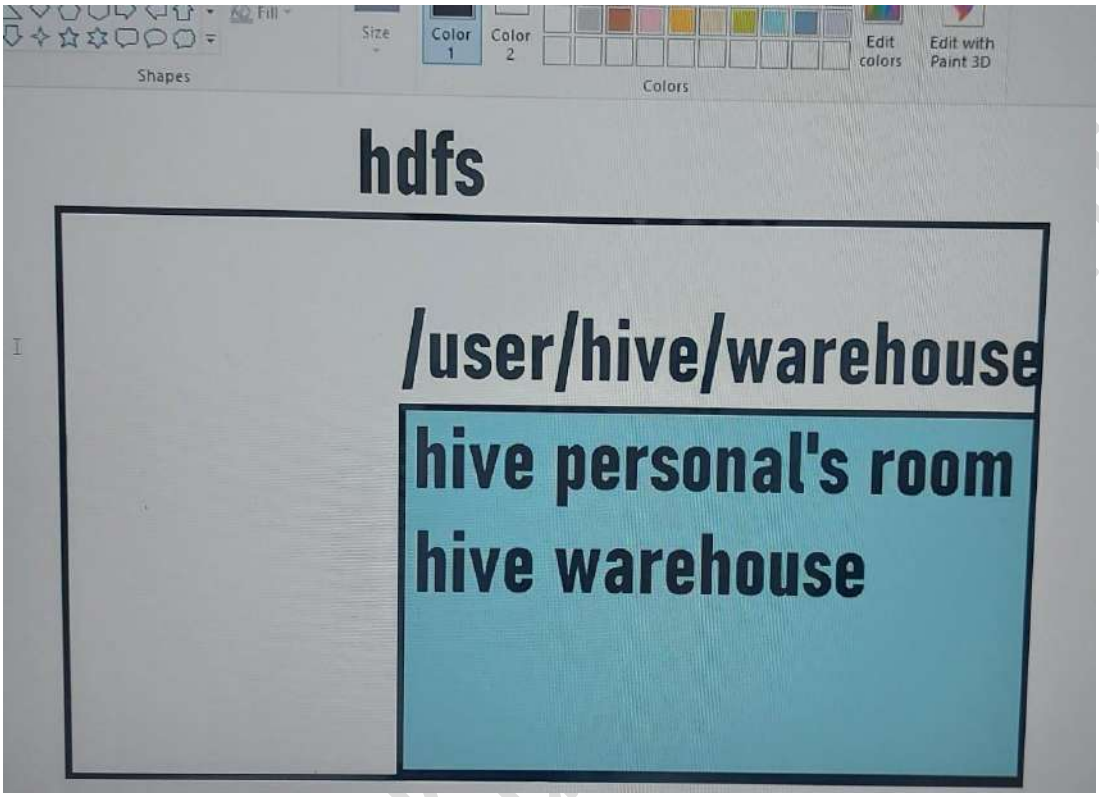
Hive is Honey-Bee on top of hadoop



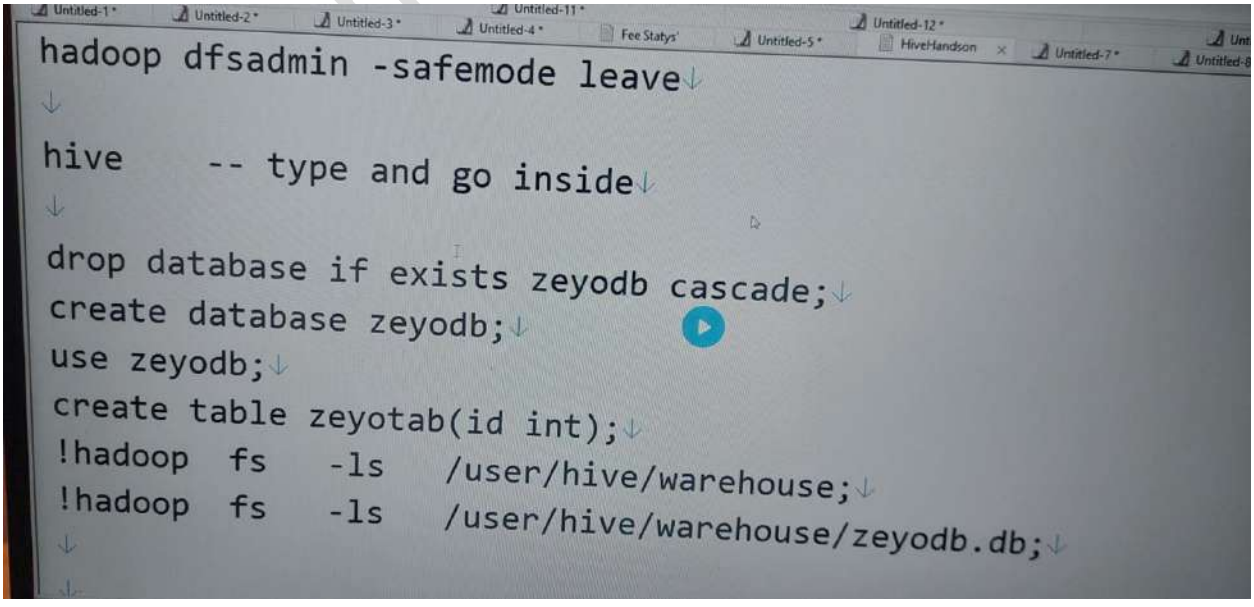
For every action on Hive, HDFS is getting impacted.

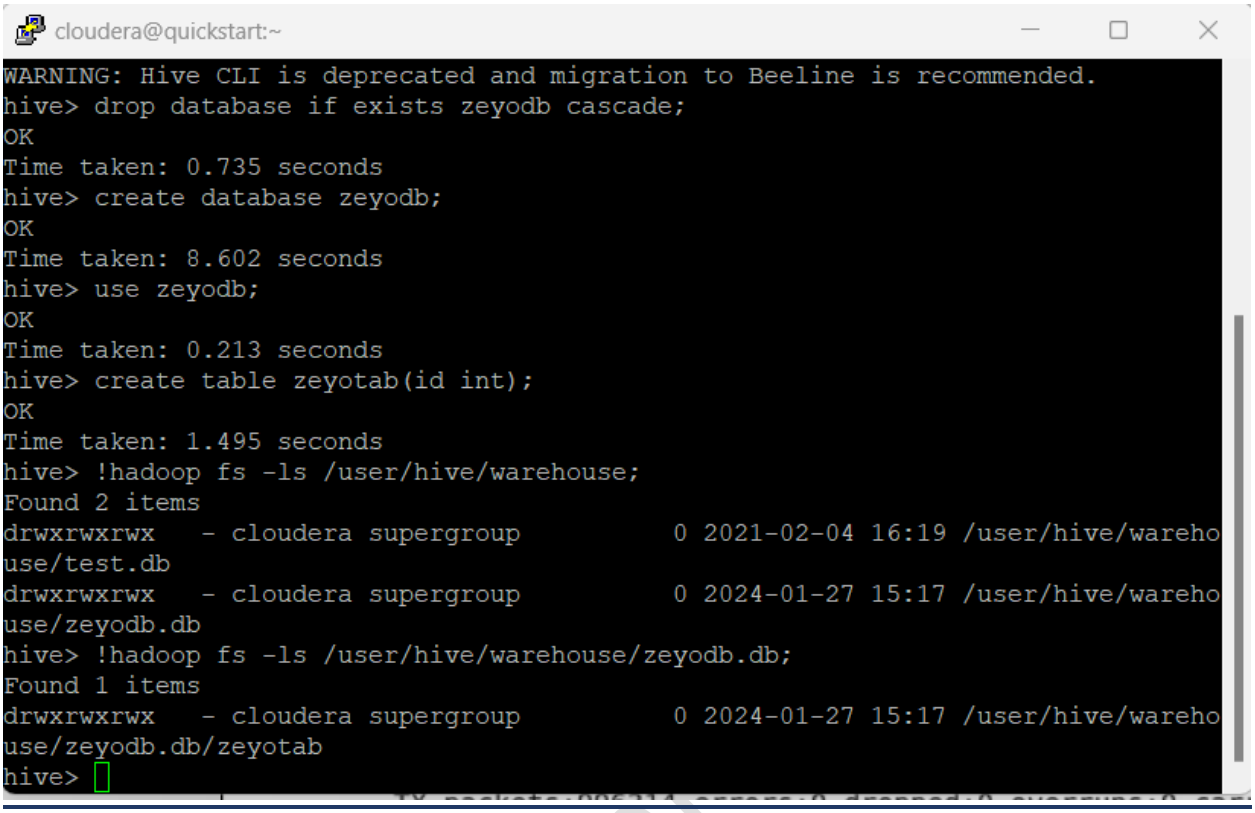
Hive Warehouse (Personal room – Dedicated Directory)

A portion of hdfs is dedicated for Hive. Directory of Hive personal room is /user/hive/warehouse



Hive DB and Table creation handson





Hive Edge node load's location

Agenda Day 16

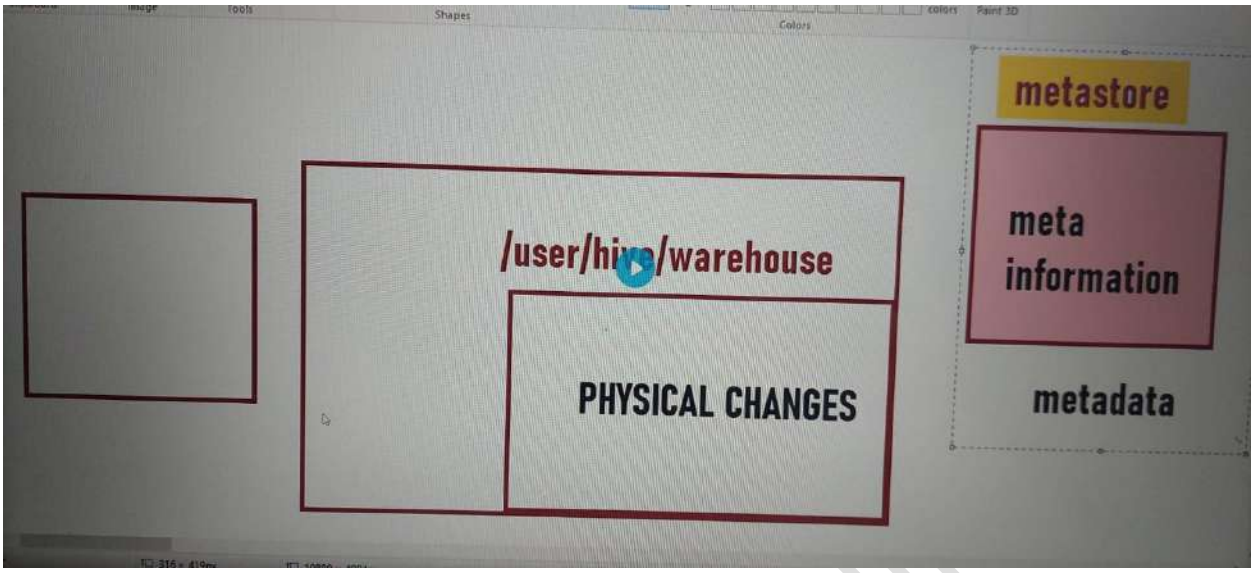
Hive Table Folder relationships

Hive runs on file System.

If we create a database in hive, a directory in the name of database gets created in Datawarehouse.

metastore – is something that contains meta information(metadata)

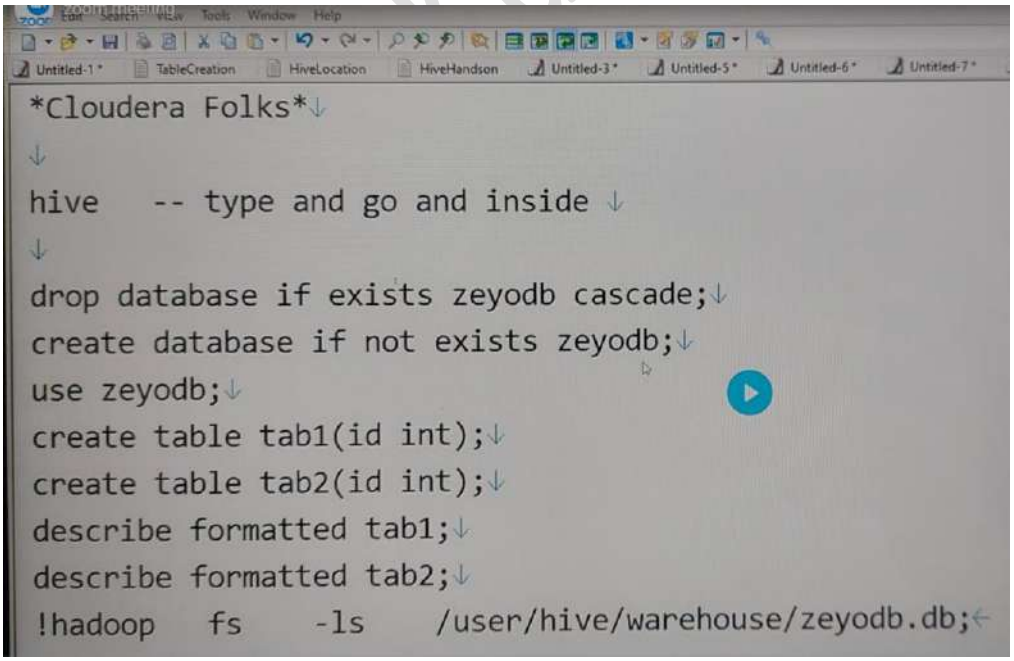
Hive directory will have physical changes.



Every Hive table must be associated/pointed to a HDFS Directory.
If we don't give any directory in hive query, hive creates its own directory in the name of table(at hive home location) and gets tagged/pointed.

HQL query to describe a table: describe formatted <table_name>

Execution:



Edge Node Loads

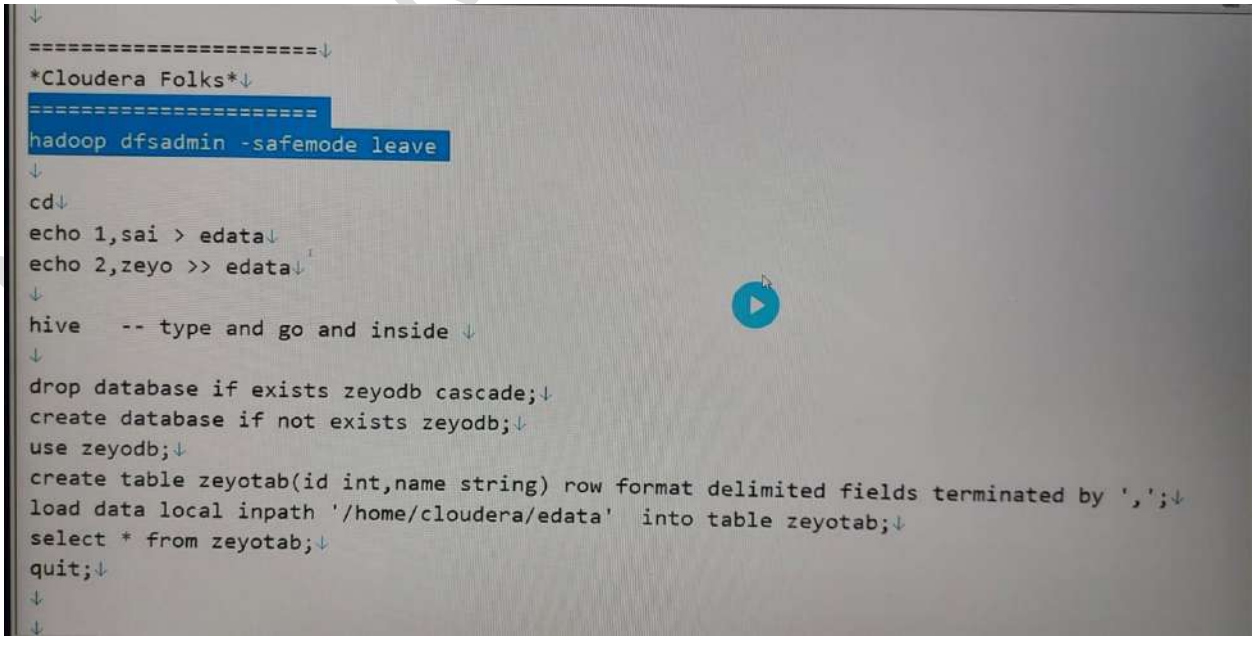
Requirement:
We have data in edge node.Go to Hive shell, Create a database ldb and use it, Create a table ltab with below details(name: ltab, columns – id(int), name string, Delimiter - Comma). Once a table is created load edge node data to this table.

Queries to be executed:

- Open Hive
- Create database ldb;
- Create table ltab(id int, name String) row format delimited fields terminated by ‘,’; (this query is to specify the incoming data is , delimited)
- Command to load the data from the Edge node to latb
  load data local inpath ‘/home/cloudera/edata’ into table ltab;
- Now the data got loaded to table ltab, we can query the table
  select * from ltab;

  Hive Commands to Enable headers
  set hive.cli.print.header = true;

Hands-on:

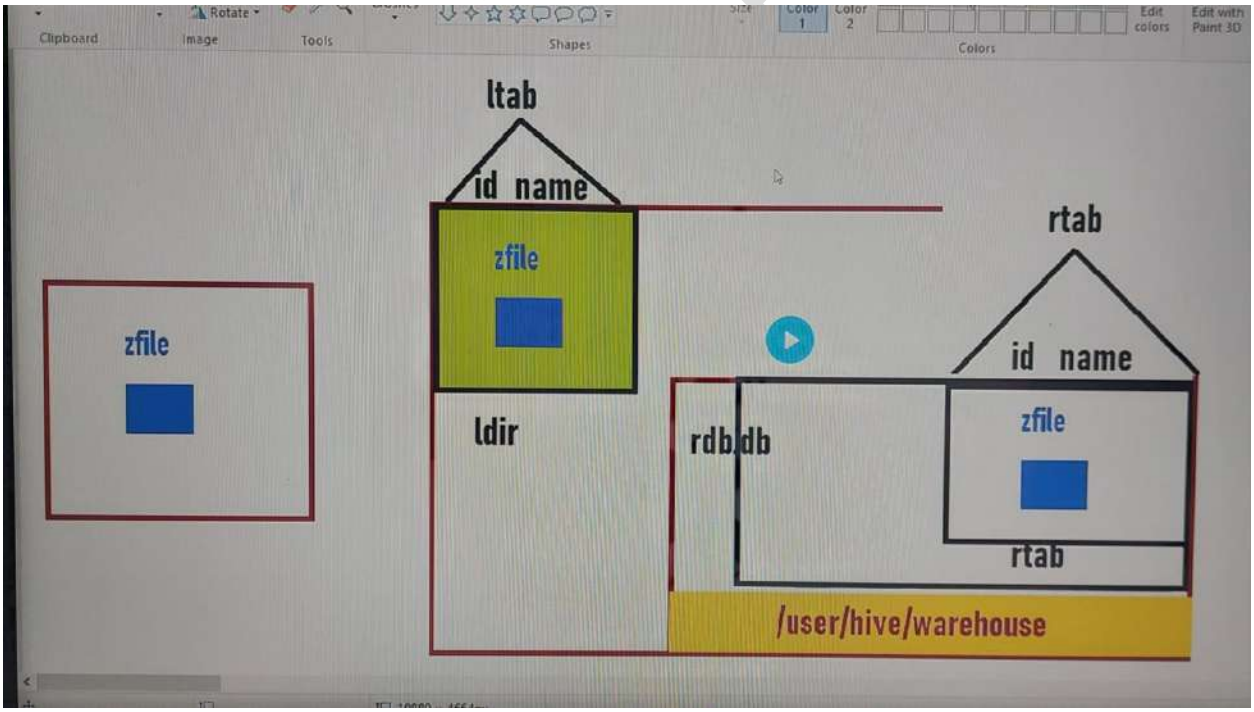


Spark Types of Tables Partitions Intro
Agenda Day 17

Location

Requirement:
Create a Hive table in a specific path.
HQL Query:
create table ltab(id int, name String) row format delimited fields terminated by ‘,’ location ‘/user/cloudera/ldir’;

Note: the required data is already loaded to the directory /user/cloudera/ldir on which hive is sitting and creating a table on top of that directory data.



Types of Tables

When a Hive Table is created on top of a directory and when we drop that table(drop table <table_name>), the directory will also gets created.

When dropping a table, whether the directory gets deleted or not completely depends on the type of table that we have created.

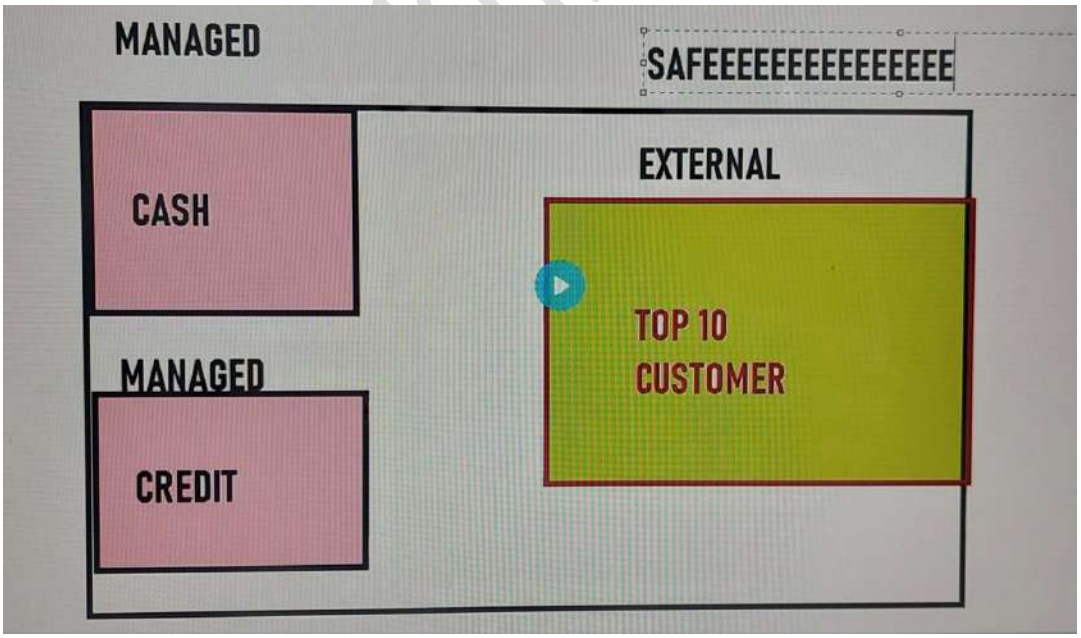
if we Created a **Managed_Table** → if we drop the table, directory will get deleted. How do we know, what is type of table we created. (describe formatted)

if we create **External Table** → even if we drop table, directory will not be deleted.

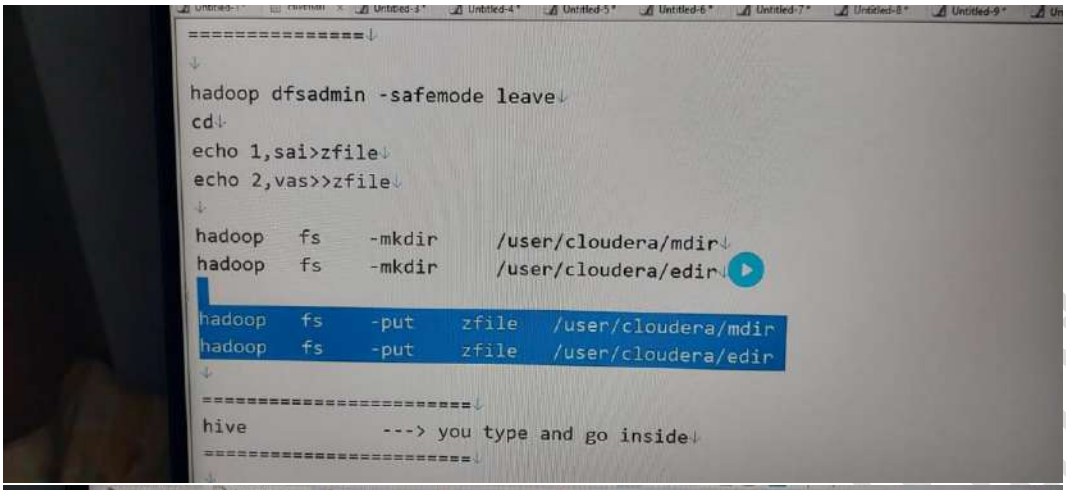
Hive – Create External Table command:

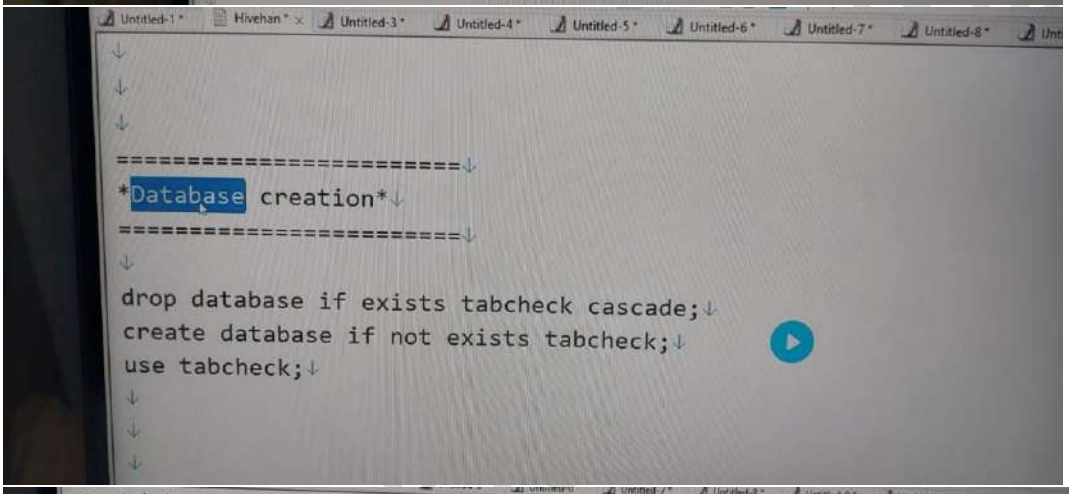
create external table if not exists <table_name>

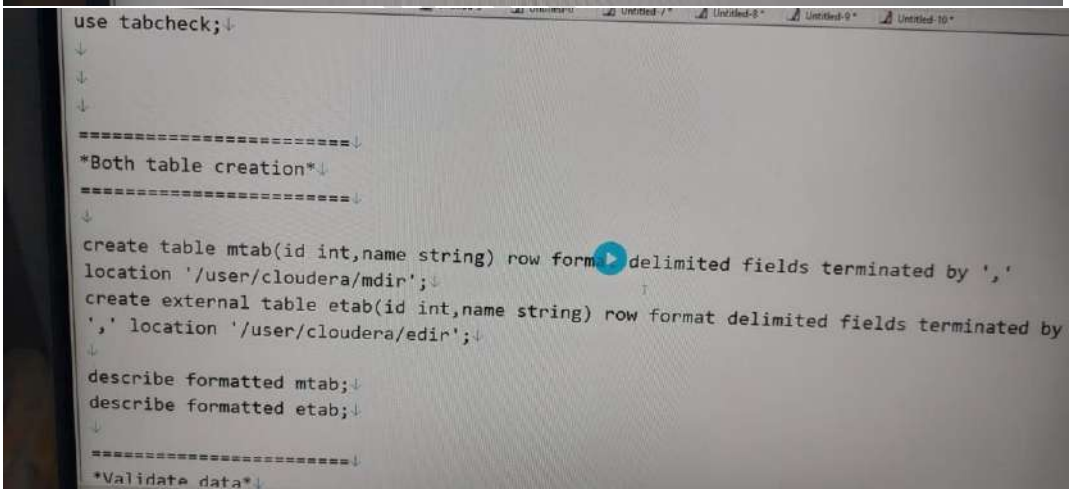
When to create Which Tables, Processing Tables should be Managed and Target tables has to be External.

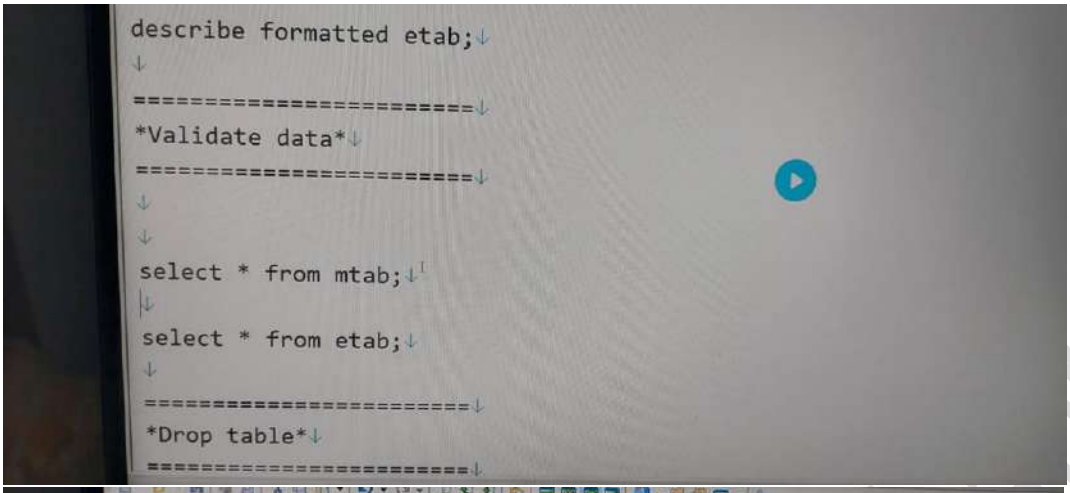


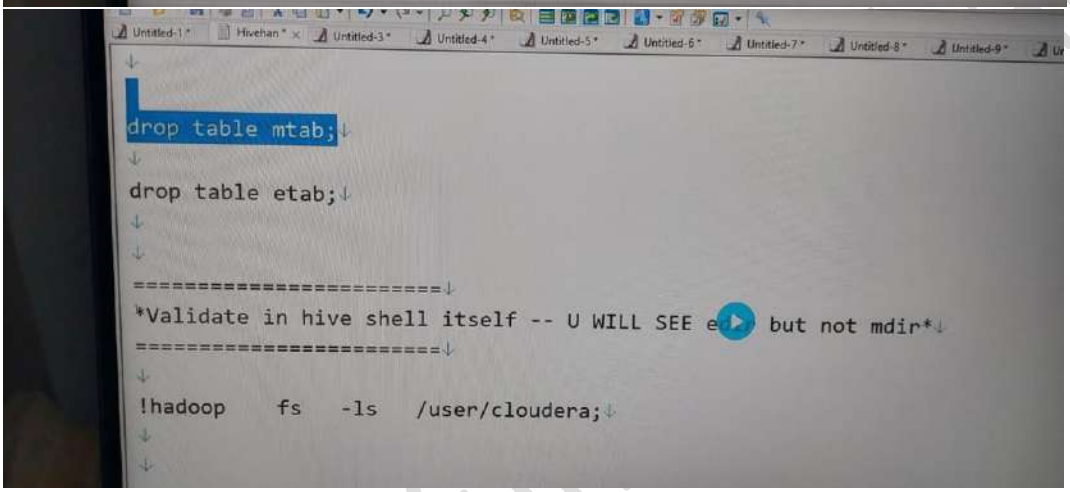
Hands-on:


==========
hadoop dfsadmin -safemode leave
cd
echo 1,sai>zfile
echo 2,vas>>zfile
hadoop fs -mkdir /user/cloudera/mdir
hadoop fs -mkdir /user/cloudera/edir
hadoop fs -put zfile /user/cloudera/mdir
hadoop fs -put zfile /user/cloudera/edir
==========
hive ---> you type and go inside
==========


==========
Database creation
==========
drop database if exists tabcheck cascade;
create database if not exists tabcheck;
use tabcheck;

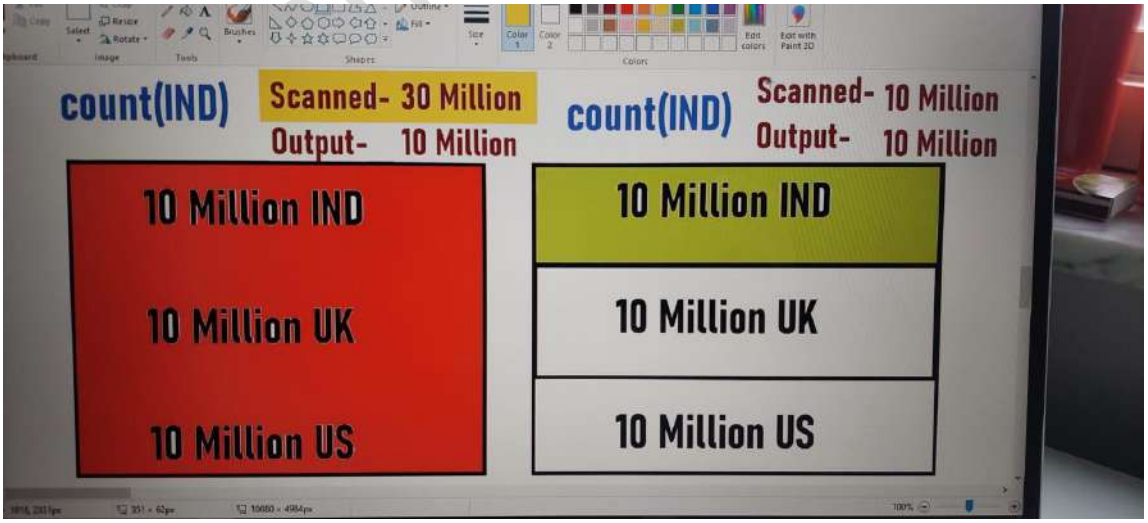

use tabcheck;
==========
Both table creation
==========
create table mtab(id int,name string) row format delimited fields terminated by ','
location '/user/cloudera/mdir';
create external table etab(id int,name string) row format delimited fields terminated by
',' location '/user/cloudera/edir';
describe formatted mtab;
describe formatted etab;
==========
Validate data





Partitions Intro

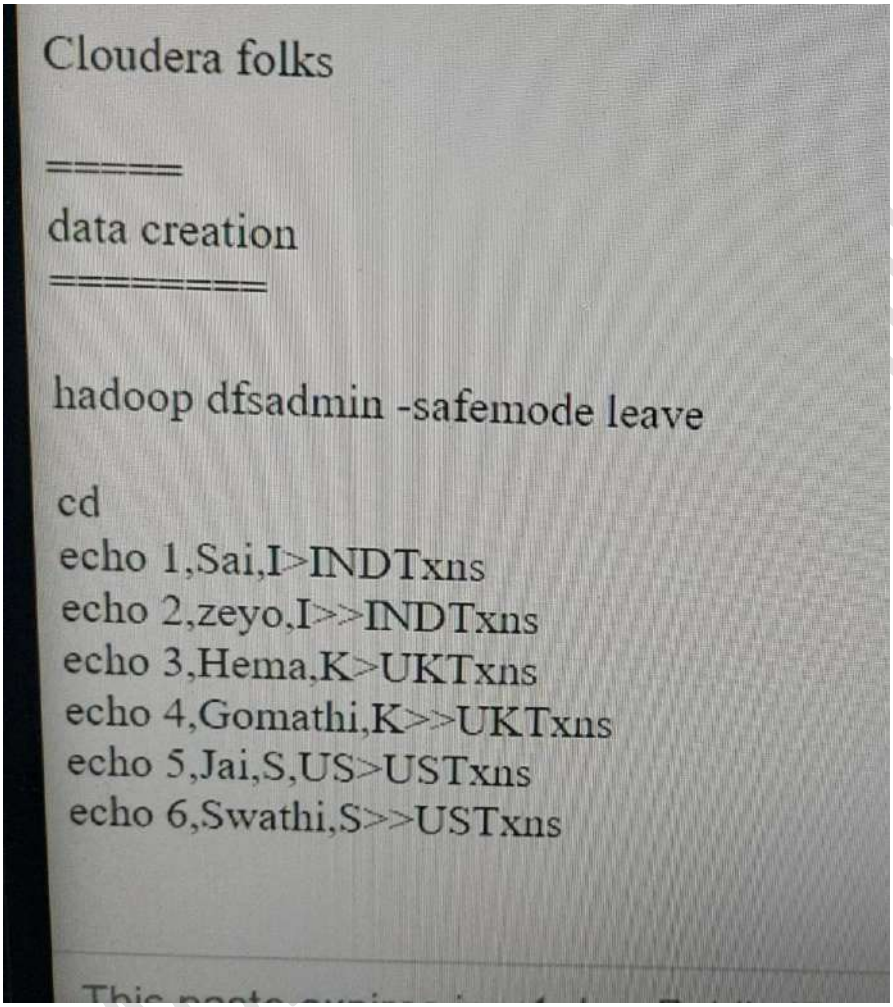
Partitions are Faster and Higher performance.(Data is partitioned on the ride side)

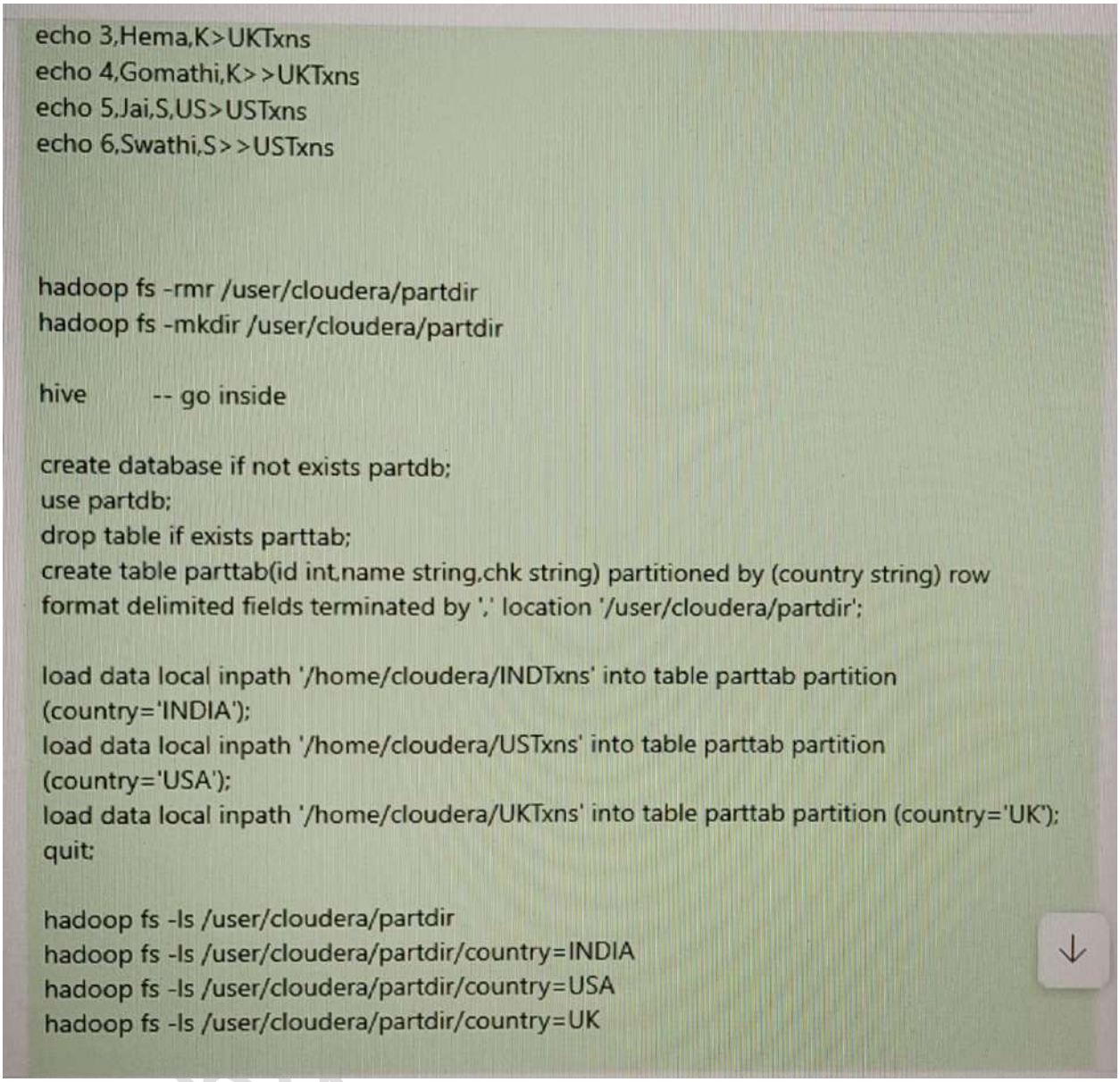


Advantages of partitions: Take only required Data.

Hands-on:

Create table parttab(id int, name string, chk string) partitioned by (country string) row format delimited fields terminated by ‘,’ location ‘/user/cloudera/partdir’;





```
echo 3,Hema,K>UKTxns
echo 4,Gomathi,K>>UKTxns
echo 5,Jai,S,US>USTxns
echo 6,Swathi,S>>USTxns



hadoop fs -rmr /user/cloudera/partdir
hadoop fs -mkdir /user/cloudera/partdir

hive        -- go inside

create database if not exists partdb;
use partdb;
drop table if exists parttab;
create table parttab(id int,name string,chk string) partitioned by (country string) row
format delimited fields terminated by ',' location '/user/cloudera/partdir';

load data local inpath '/home/cloudera/INDTxns' into table parttab partition
(country='INDIA');
load data local inpath '/home/cloudera/USTxns' into table parttab partition
(country='USA');
load data local inpath '/home/cloudera/UKTxns' into table parttab partition (country='UK');
quit;

hadoop fs -ls /user/cloudera/partdir
hadoop fs -ls /user/cloudera/partdir/country=INDIA
hadoop fs -ls /user/cloudera/partdir/country=USA
hadoop fs -ls /user/cloudera/partdir/country=UK
```

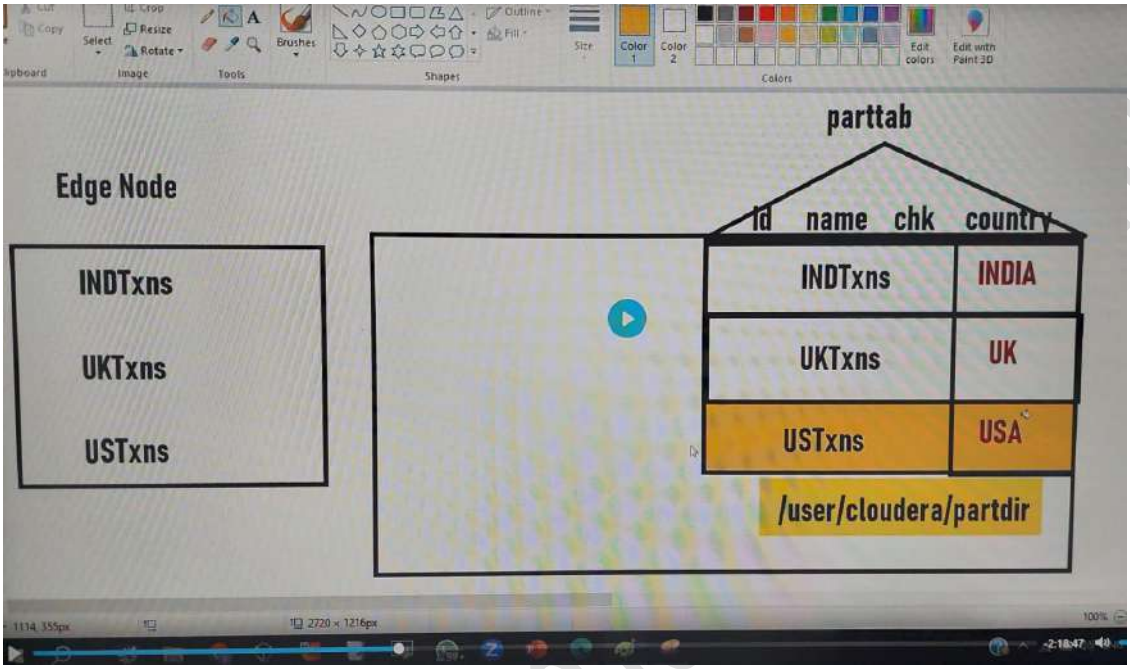
Partioned by (country string) → creates new column, when you describe table, we will see four columns if we execute the above command and also partition information by country.

load INDTxns with parttion name as ‘INDIA’ → command
load data local inpath ‘/home/cloudera/INDtxns’ into table parttab partition (country = ‘INDIA‘)

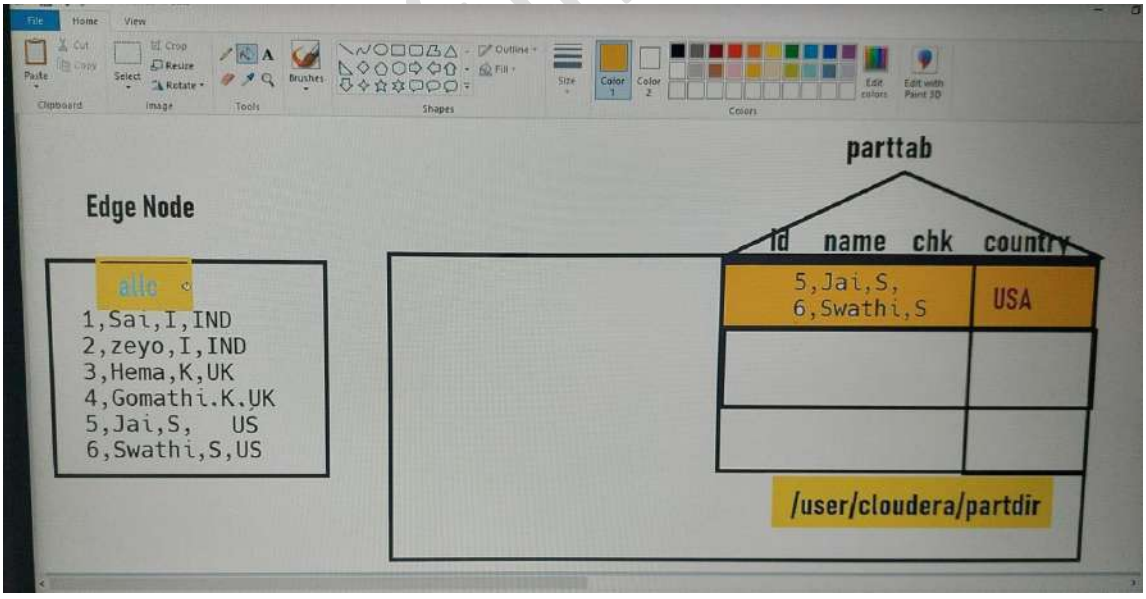
Hive Partitions, Static, Dynamic, Revision

Agenda Day-18

Static Load – We have data already divided

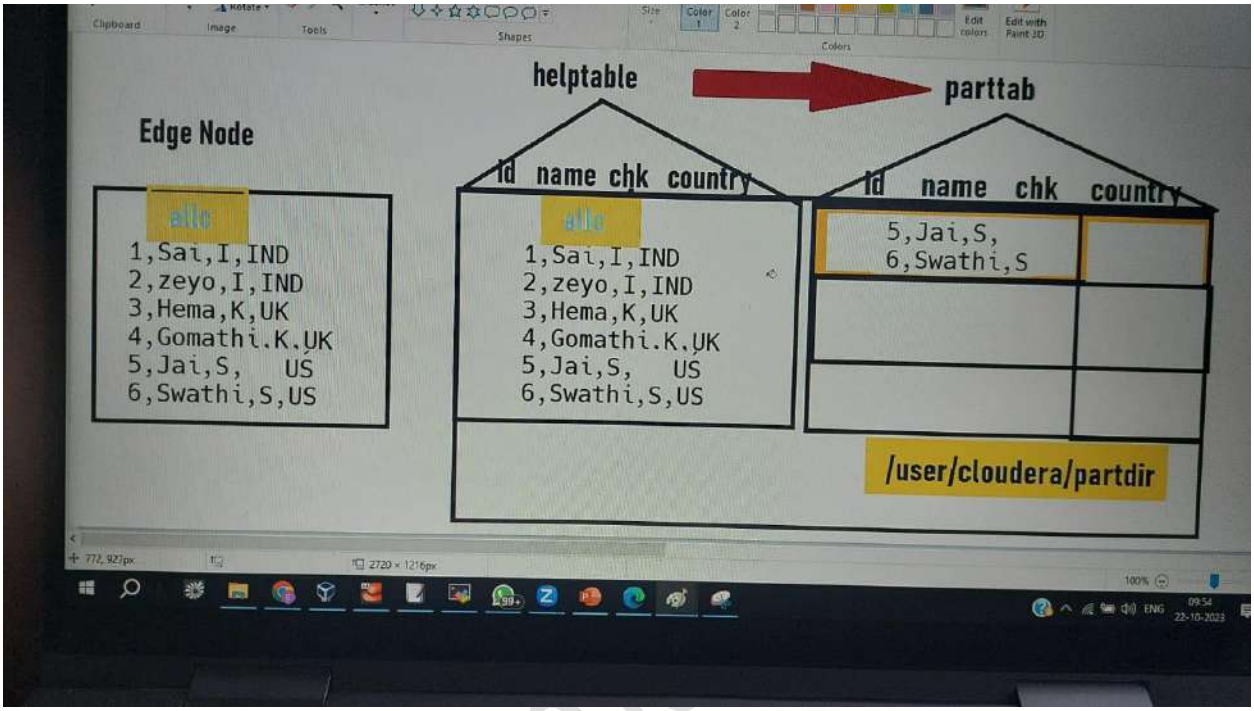


Static Insert



Here we got only a single file with all the country details, and we want load specific country records to partition.

Solution: we need one helping table for this case. Create one normal table (id, name, chk, country), load allc files to this table and then from this table filter US records alone and load to partition as USA.



Handson:

Insert into parttab partition(country='USA') select id, name, chk from helptable where chk='S';

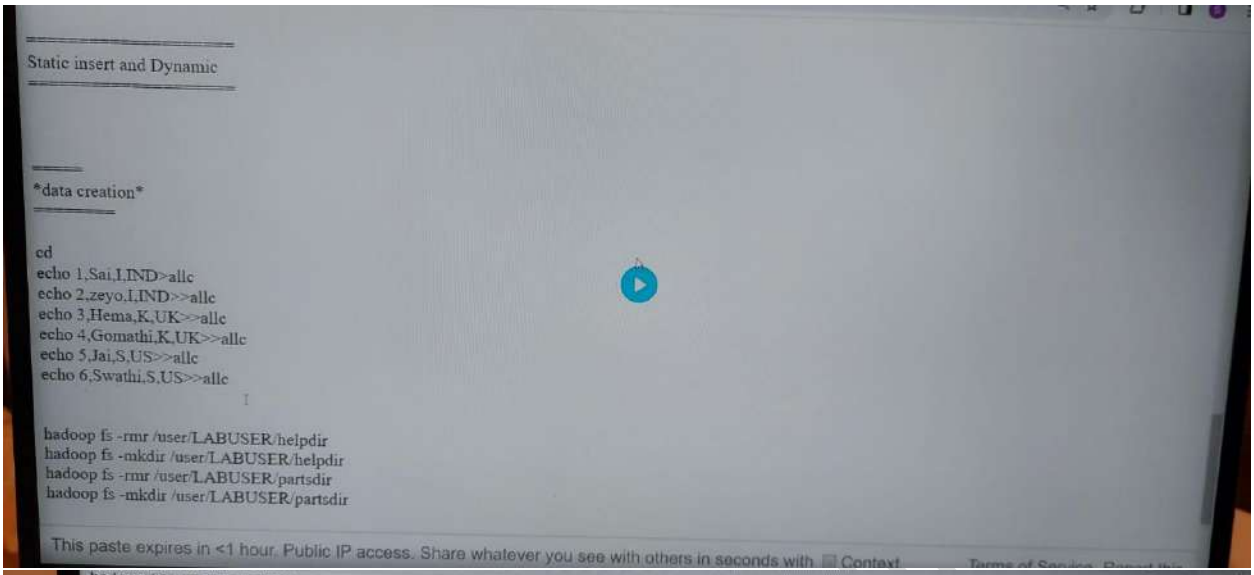
Dynamic Partition:

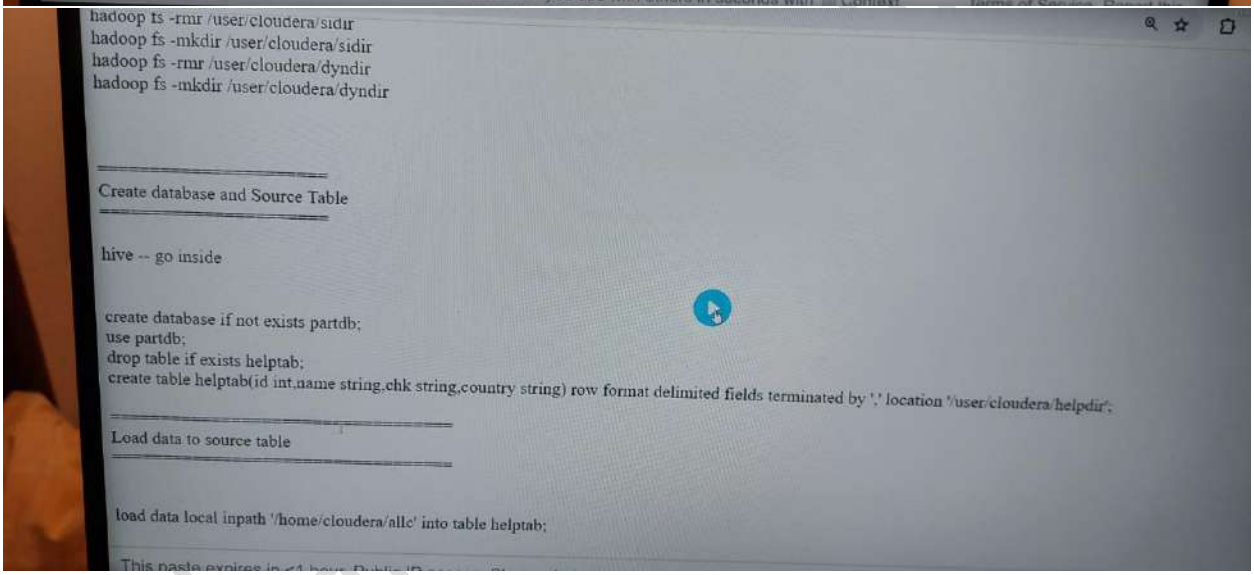
Now, based on the country the partitions to be created automatically.

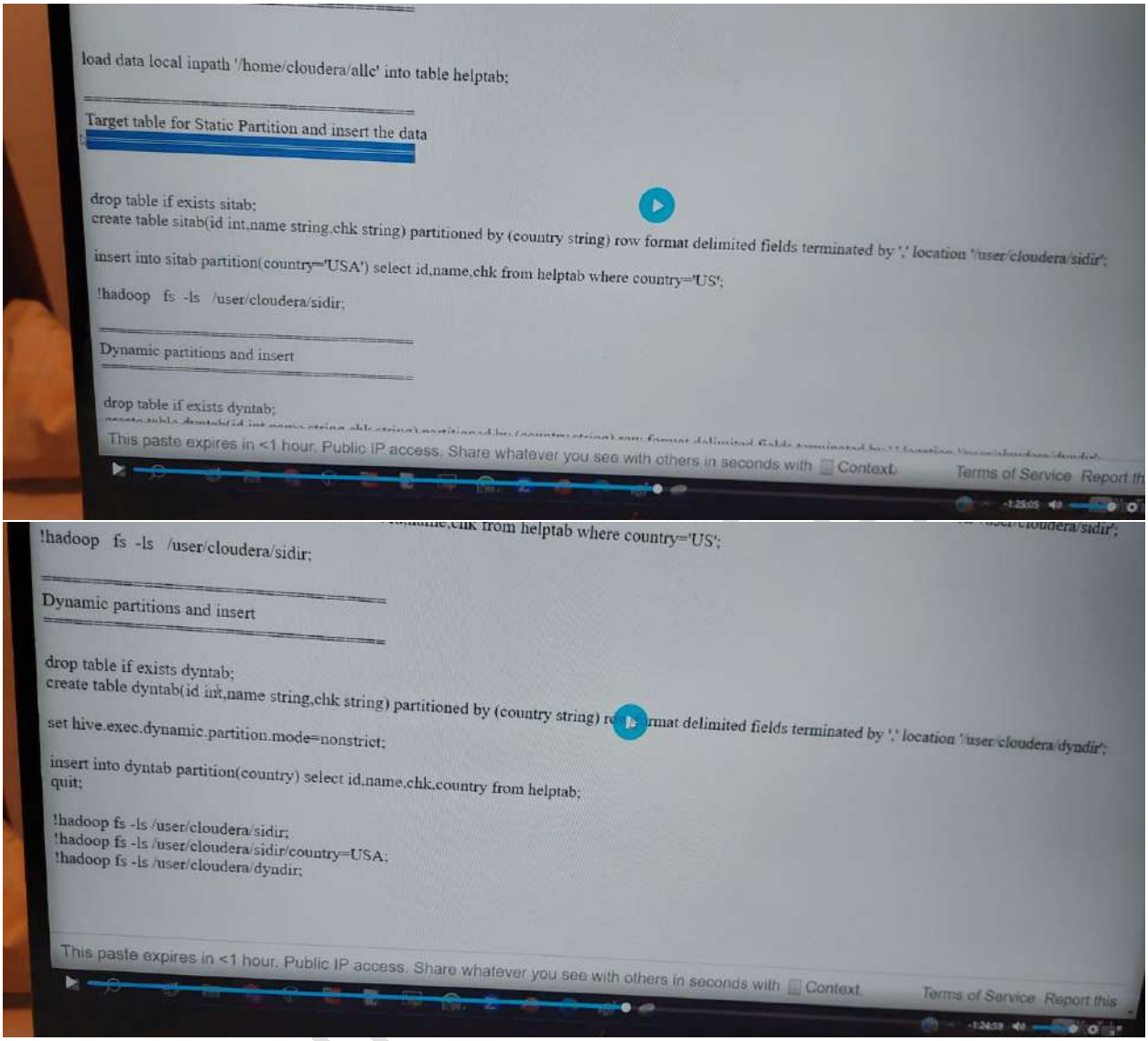
Command:

Insert into parttab partition(country) select id,name,chk,country from helptable;

Handson Static and Dynamic partition:






load data local inpath '/home/cloudera/allc' into table helptab;
Target table for Static Partition and insert the data
drop table if exists sitab;
create table sitab(id int,name string,chk string) partitioned by (country string) row format delimited fields terminated by ',' location '/user/cloudera/sidir';
insert into sitab partition(country='USA') select id,name,chk from helptab where country='US';
!hadoop fs -ls /user/cloudera/sidir;
Dynamic partitions and insert
drop table if exists dyntab;
This paste expires in <1 hour. Public IP access. Share whatever you see with others in seconds with Context.
Terms of Service Report th
!hadoop fs -ls /user/cloudera/sidir;
Dynamic partitions and insert
drop table if exists dyntab;
create table dyntab(id int,name string,chk string) partitioned by (country string)
set hive.exec.dynamic.partition.mode=nonstrict;
insert into dyntab partition(country) select id,name,chk,country from helptab;
quit;
!hadoop fs -ls /user/cloudera/sidir;
!hadoop fs -ls /user/cloudera/sidir/country=USA;
!hadoop fs -ls /user/cloudera/dyndir;
This paste expires in <1 hour. Public IP access. Share whatever you see with others in seconds with Context.
Terms of Service Report this

Hive Cloud AWS Azure GCP AVRO Schema Evolution Project Agenda Day -19

The helping table above is called Staging Table

Cloud Hive Analytics

Hadoop -Cloud

Google Cloud – Dataproc – Hadoop Managed service
Amazon – EMR (Elastic Map Reduce)
Azure – HDInsight

Cloudera Hive VS Cloud Hive (AWS, AZURE,GCP)

Cloud Hive not only allows us to create tables in HDFS, it also allows us to create tables on Cloud storages.

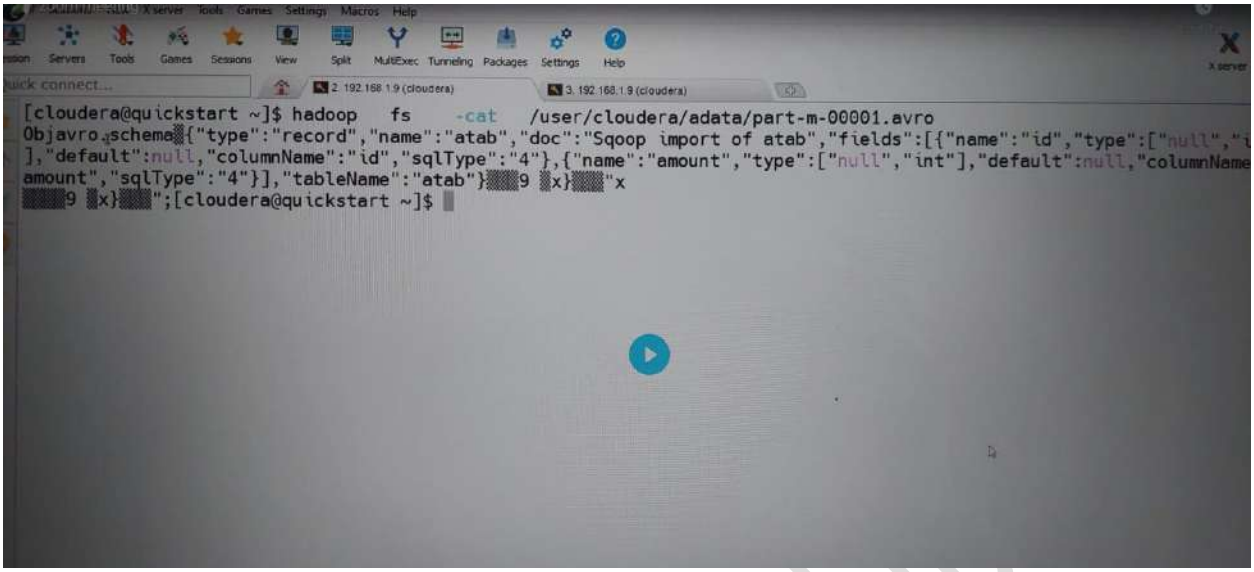
Example: create table ctab(name string) row format delimited fields terminated by ‘,’ location “gs://zeyobuck/zeyodir/”;

gs://zeyobuck/zeyodir/ → google cloud Storage location

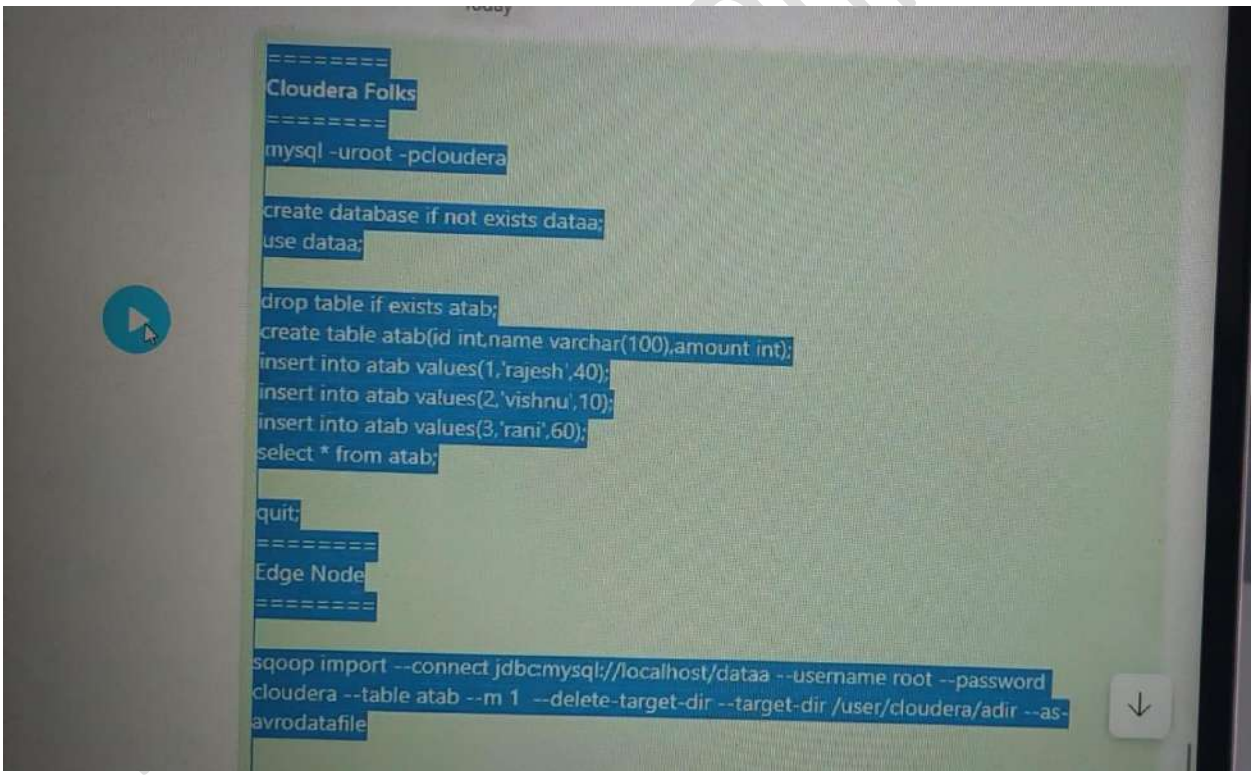
AWS → EMR – S3
Azure → hdinsight – blob
GCP → dataproc – Cloud Storage

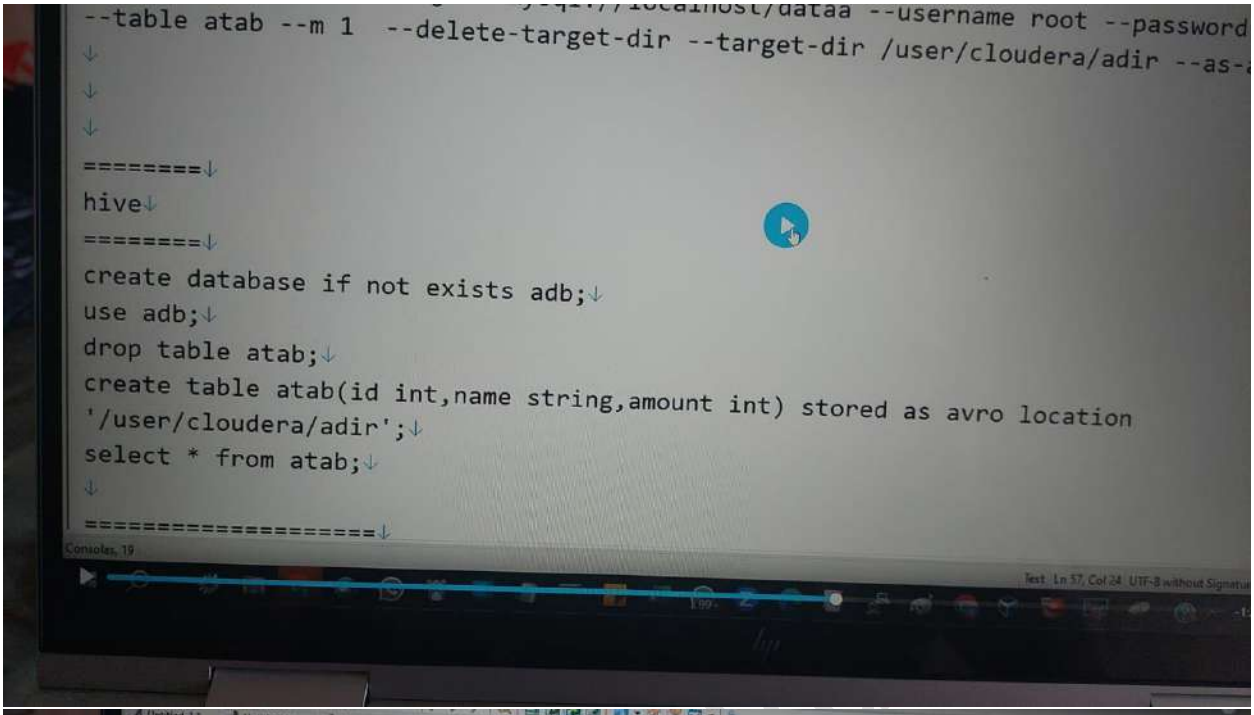
AVRO Schema Evolution

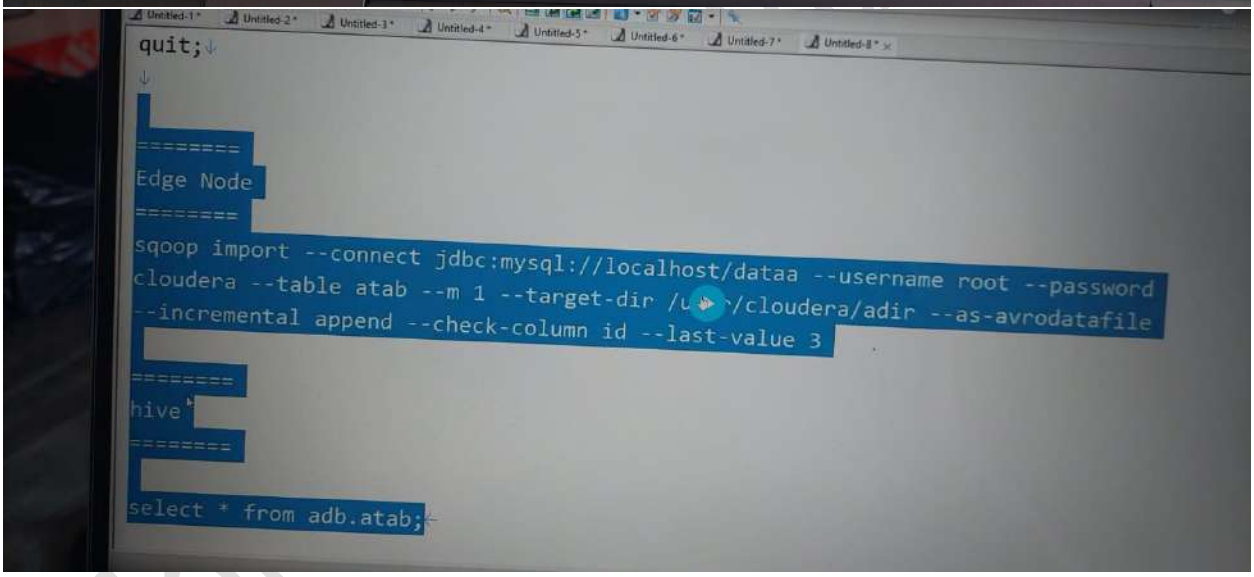
Schema – Columns
Evolution - Change
avro file will bring the columns as well. In Avro, we will have two sections [JSON Columns and Serialized data]


[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/adata/part-m-00001.avro
Objavro.schema{"type":"record","name":"atab","doc":"Sqoop import of atab","fields":[{"name":"id","type":["null","int"],"default":null,"columnName":"id","sqlType":"4"},{"name":"amount","type":["null","int"],"default":null,"columnName":"amount","sqlType":"4"}],"tableName":"atab"}9 x}"x
9 x}";[cloudera@quickstart ~]$

Execution

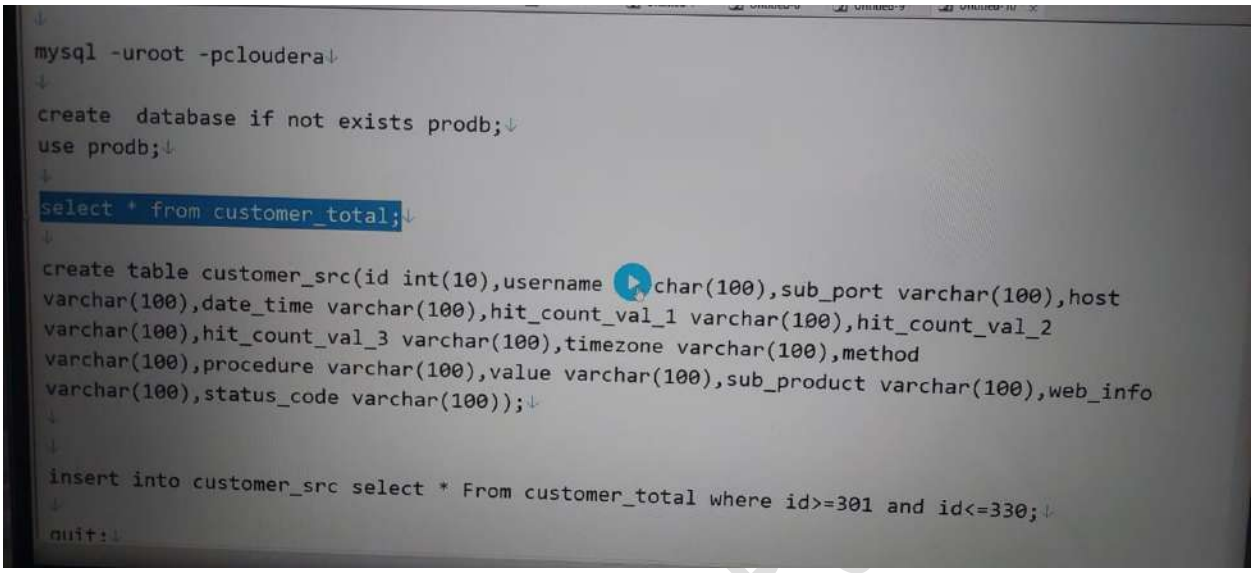

========
Cloudera Folks
========
mysql -uroot -pcloudera
create database if not exists dataa;
use dataa;
drop table if exists atab;
create table atab(id int,name varchar(100),amount int);
insert into atab values(1,'rajesh',40);
insert into atab values(2,'vishnu',10);
insert into atab values(3,'rani',60);
select * from atab;
quit;
========
Edge Node
========
sqoop import --connect jdbc:mysql://localhost/dataa --username root --password cloudera --table atab --m 1 --delete-target-dir --target-dir /user/cloudera/adir --as-avrodatafile


--table atab --m 1 --delete-target-dir --target-dir /user/cloudera/adir --as-
========
hive
========
create database if not exists adb;
use adb;
drop table atab;
create table atab(id int,name string,amount int) stored as avro location
'/user/cloudera/adir';
select * from atab;
====================


quit;
========
Edge Node
========
sqoop import --connect jdbc:mysql://localhost/dataa --username root --password
cloudera --table atab --m 1 --target-dir /u
--incremental append --check-column id --last-value 3
========
hive
========
select * from adb.atab;

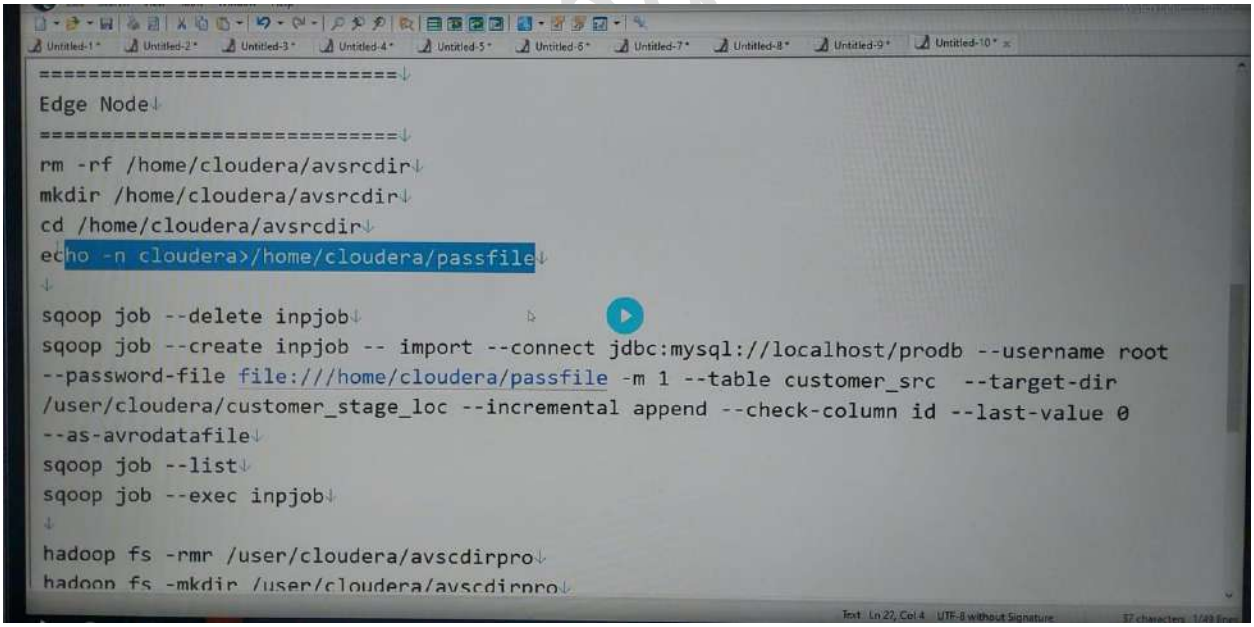
Phase 1 Project

Customer_total table will be there in Cloudera RDBMS table already. Create table customer_src and insert some records from customer_total table.



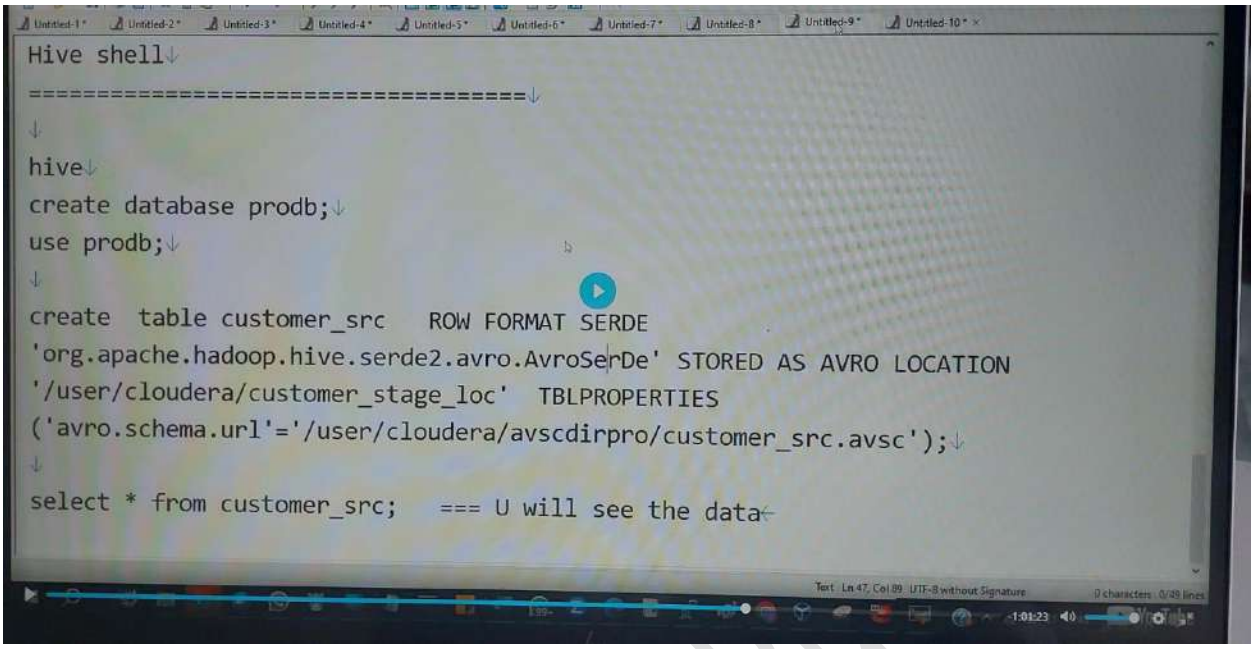
```
mysql -uroot -pcloudera

create  database if not exists prodb;
use prodb;

select * from customer_total;

create table customer_src(id int(10),username char(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>=301 and id<=330;

quit;
```

Create a Sqoop Automation Job with Password file with AVRO and execute the job and get the Customer_src data to HDFS



```
==============================
Edge Node
==============================
rm -rf /home/cloudera/avsrcdir
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 --list
sqoop job --exec inpjob

hadoop fs -rmr /user/cloudera/avscdirpro
hadoop fs -mkdir /user/cloudera/avscdirpro
```

Open Hive Shell and Create Hive table on top of it



Observation:

You will Read the AVRO Schema from a file????

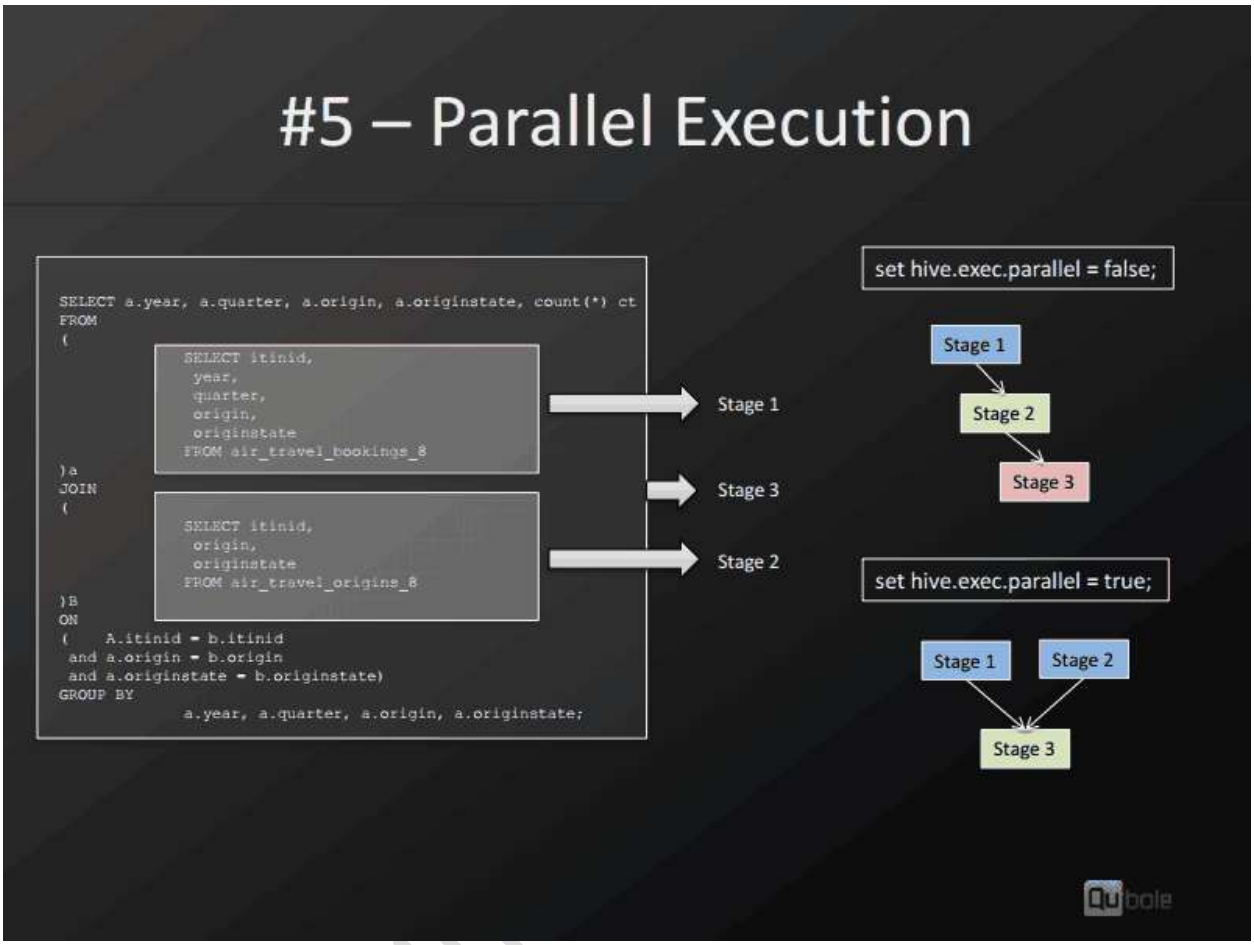
Hive Performance Tuning, Spark Intro, MR vs Spark

Agenda Day 20

Hive Performance Tuning

- Partitions Improve the performance.
- Parquet file format
- Enabling Parallel Execution

  set hive.execution.parallel =true;



Parallel execution needs high computational power, so By default it would be set to false.

To look the current values(execute below commands in hive):

set hive.exec.parallel → shows what value is set currently

set hive.exec.parallel = true;

- Engine Change (Execute below command in hive)
  set hive.execution.engine → gives us what engine is running at the backend
  set hive.execution.engine = tez (tez is much faster, if MR takes 10min, tez takes 30-40 sec) why tez is faster? It's proportional to spark.
- Vectorization in hive
  set hive.vectorized.execution.enabled;
  set hive.vectorized.execution.enabled = true;
  By default Hive process 1 row at a time, By enabling Vectorization – hive

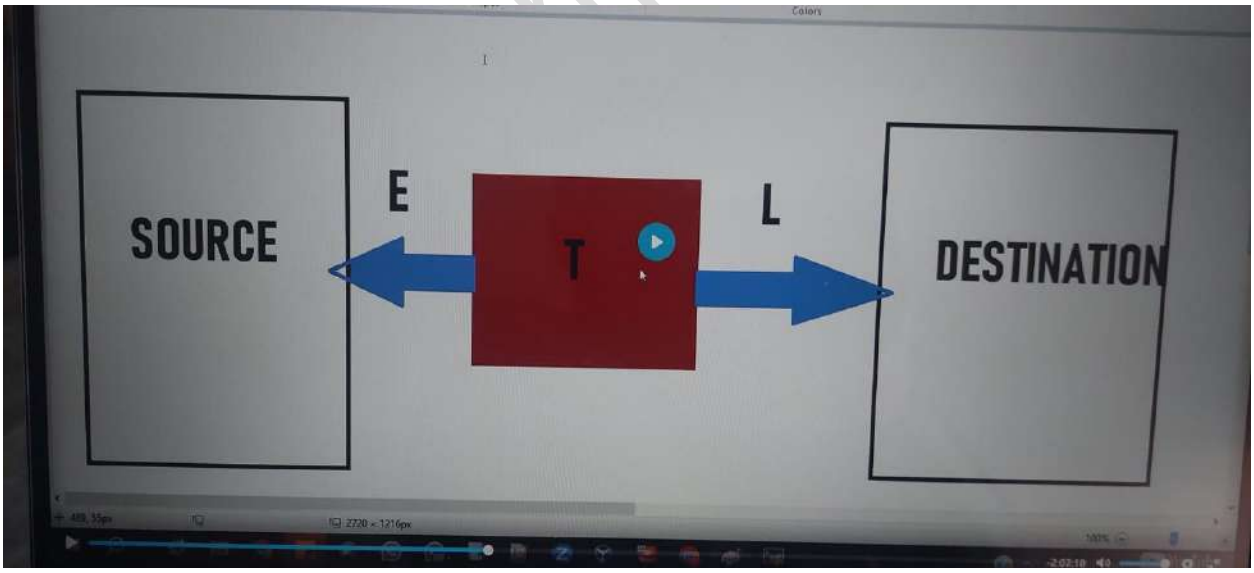
process 1024 rows at a time.

Spark Intro and Evolution

Hive Restrictions (By 2012):

- Environment dependent, it processes data only in HDFS. (won't support non-hdfs environments)
- No Support to streaming, ML Support, Entire ETL – No (only supports Transformation in HDFS), Customization, Other than MR Engine.
- It can handle MB, GB, TB but not PG,EB,ZB,YB,BB,GB.

2012 – Matei Zaharia --- Came up with unified solution.
Features:
He introduced a tool named spark which is simple and faster, Environment Independent, Very good Streaming Support, good ML Support, good SQL Support. Customization is easy and ETL is possible with one formulae and with one line of code.

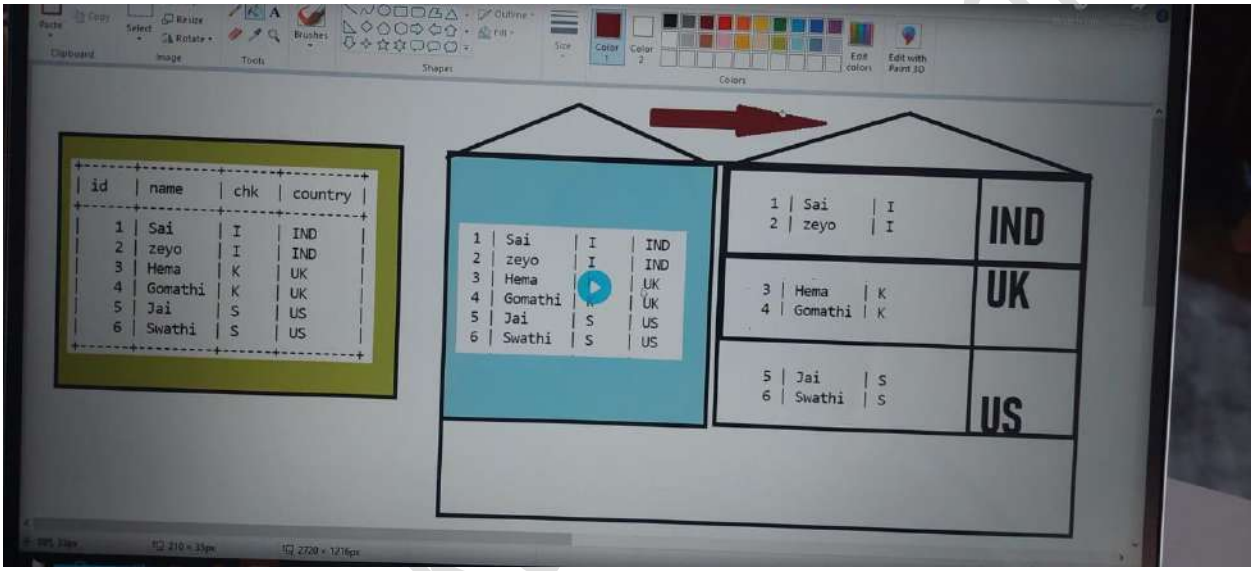


SPARK tagline is Lightening Faster Cluster Computing

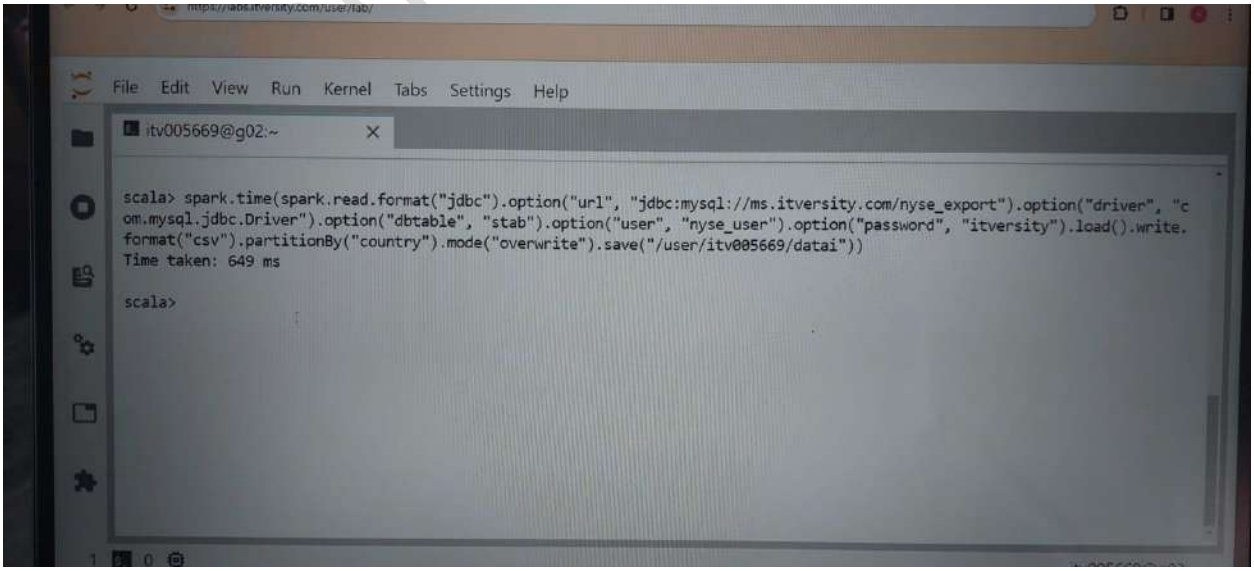
Spark Sample Execution

Normal Execution (Takes min 2min Using Sqoop and Hive):

Execute Sqoop import command first
Create external table with partioned column with country
Create staging (help) table on sqoop import target table
Enable Dynamic Partitions
Do Table to Table insert with last column country



With Spark (Takes less than a sec):



Why Spark is faster.

To Understand Why Spark is faster, Lets take word count example and Compare MR vs Spark

Input:
Spark hadoop
hive spark

hive spark
hadoop hive

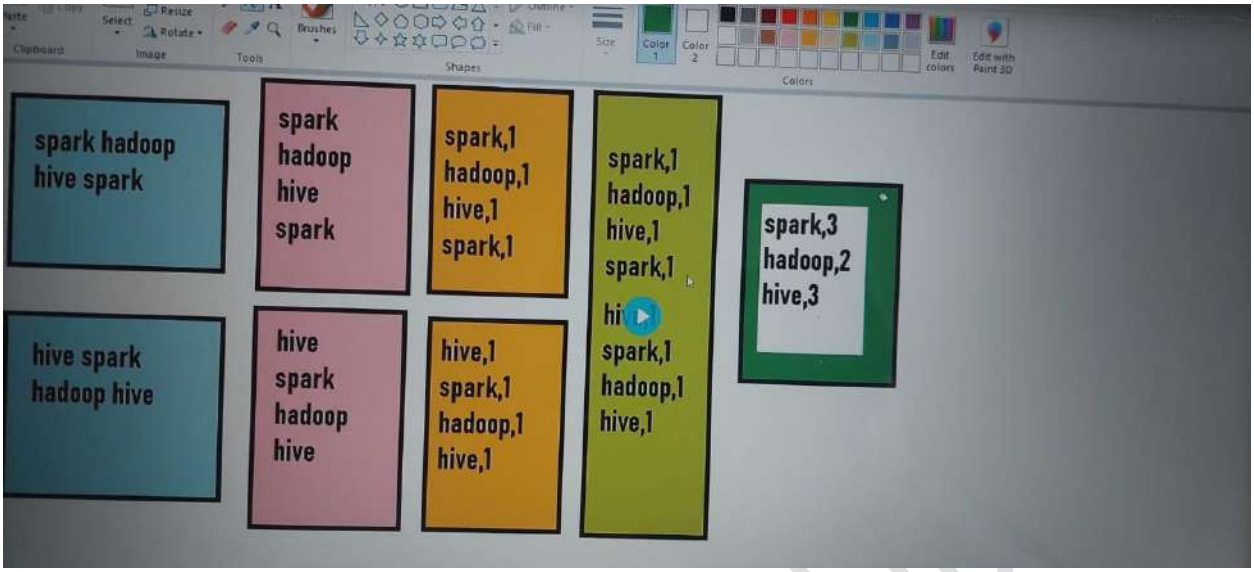
MR or Spark will do below 5 steps to do word count

1) Read the data
2) Flatten with Space

spark
Hadoop
Hive
Spark

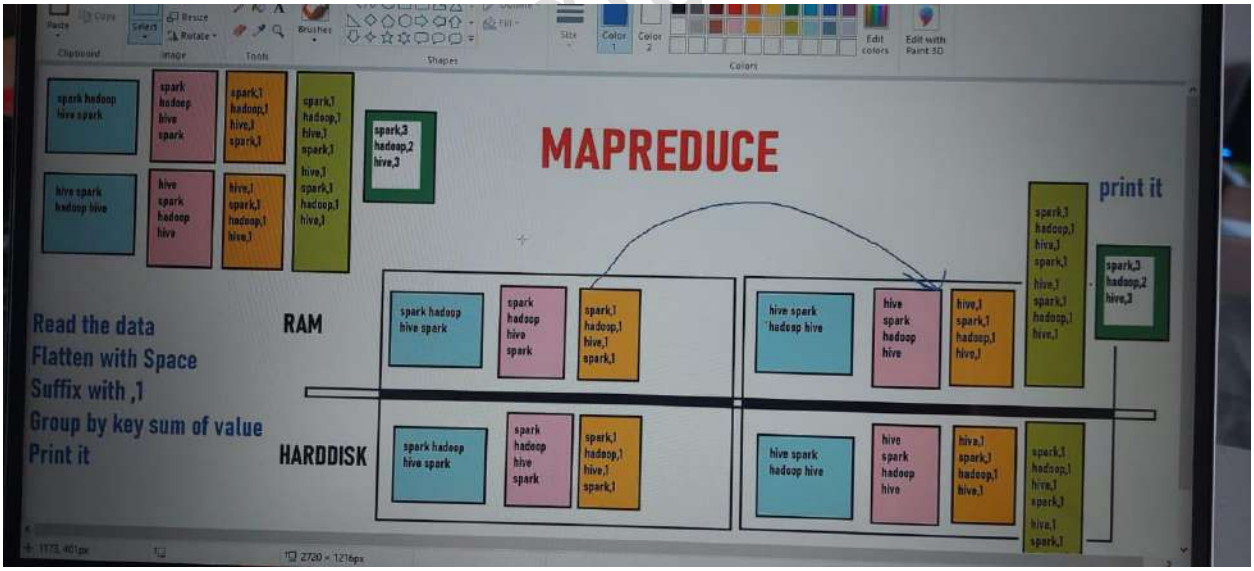
Hive
Spark
Hadoop
hive

3) Suffix with , 1
4) Group by key sum of value
5) Print it



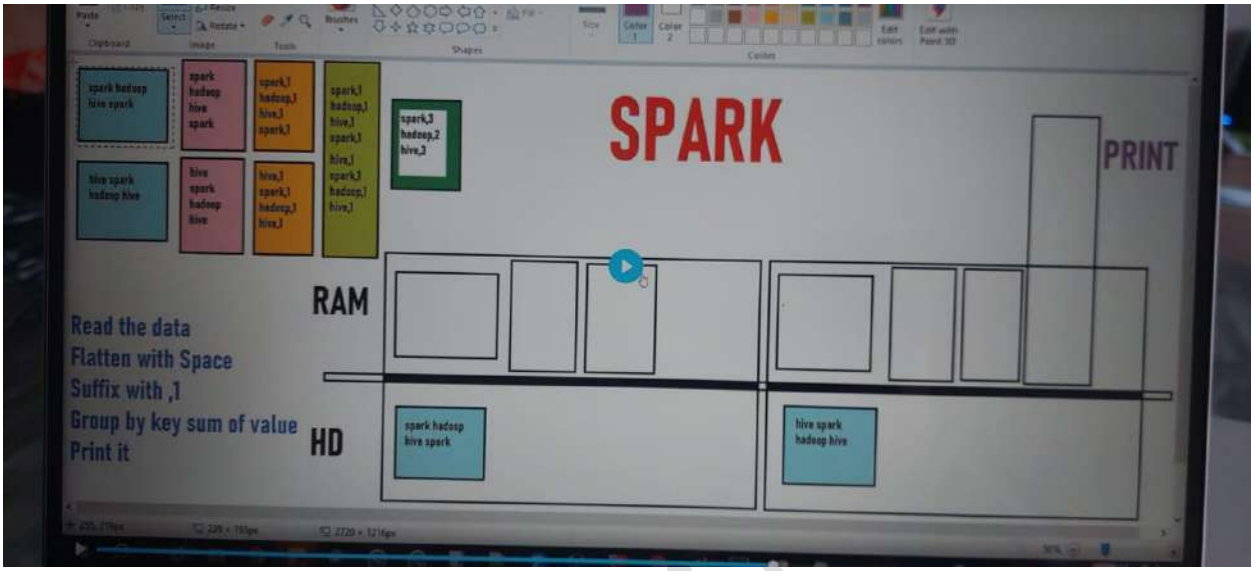
Execution with Map-reduce

Every time map-reduce hits the Hard-disk every time to store intermittent results. Because – Map reduce is linear in nature, It's not sure about next steps, No proper planning. For safety it saves intermediate data to harddisk. Storing this intermittent results causing to take lot of time.

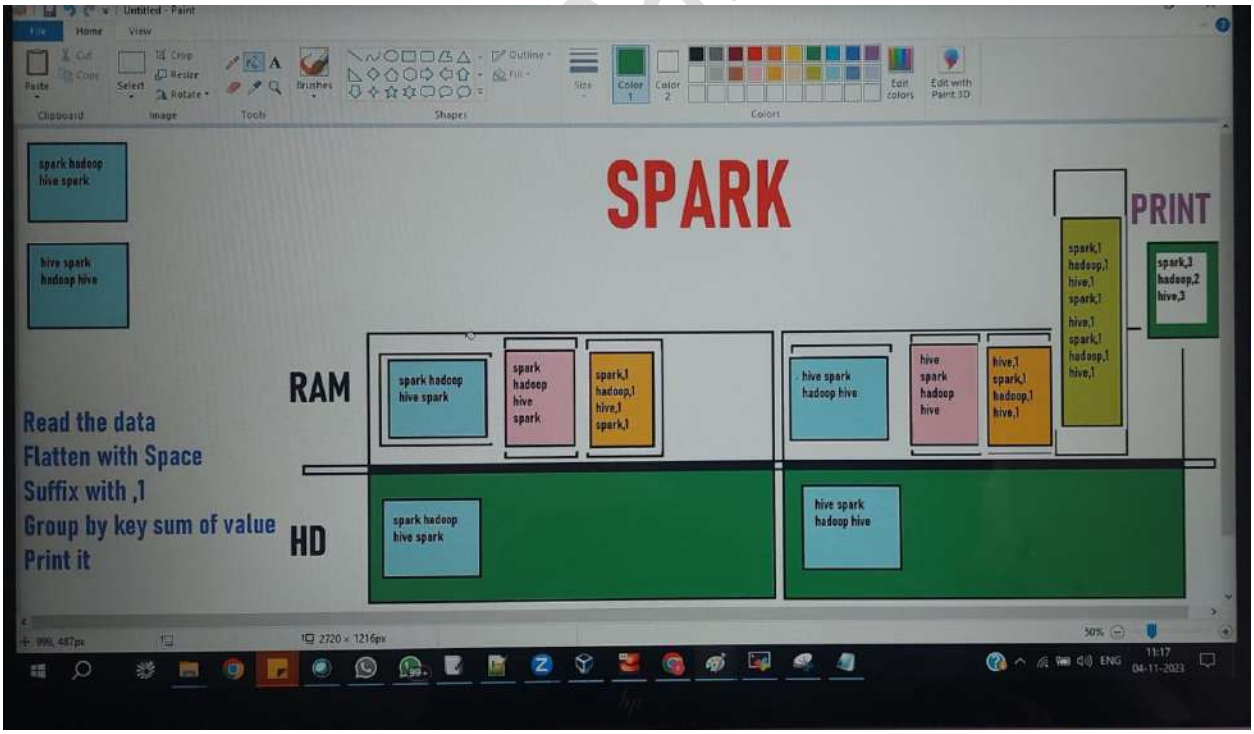


Where as spark is not linear in nature. Spark is very LAZY in nature. Till the action is triggered none of the transformations takes place, it will just plan.

Till Step-5 print (which is action is called) – it will just plan everything



After Action is Called, everything planned will be executed.



Spark follows Lazy Evaluation.

Set up Details

Tasks:

1) What is Mapside join in Hive?
2) Setup
   Eclipse: https://youtu.be/zUPbe06ZOsw
   Intellij: https://www.youtube.com/watch?v=eHk1lJ6kb14
3) Code:

```
Object obj {
    def main(args:Array[String]):Unit={
            print("==== Spark Journey Started======")
    }
}
```

Spark vs Mr Performance run, deployment, Lazy Evaluation Agenda 21:

Mapside join in Hive:

When the table data is distributed in multiple back-end nodes, to join them it would be difficult as below.

But syntax below, will bring a copy of Table B to all the nodes, so that join would be easy.

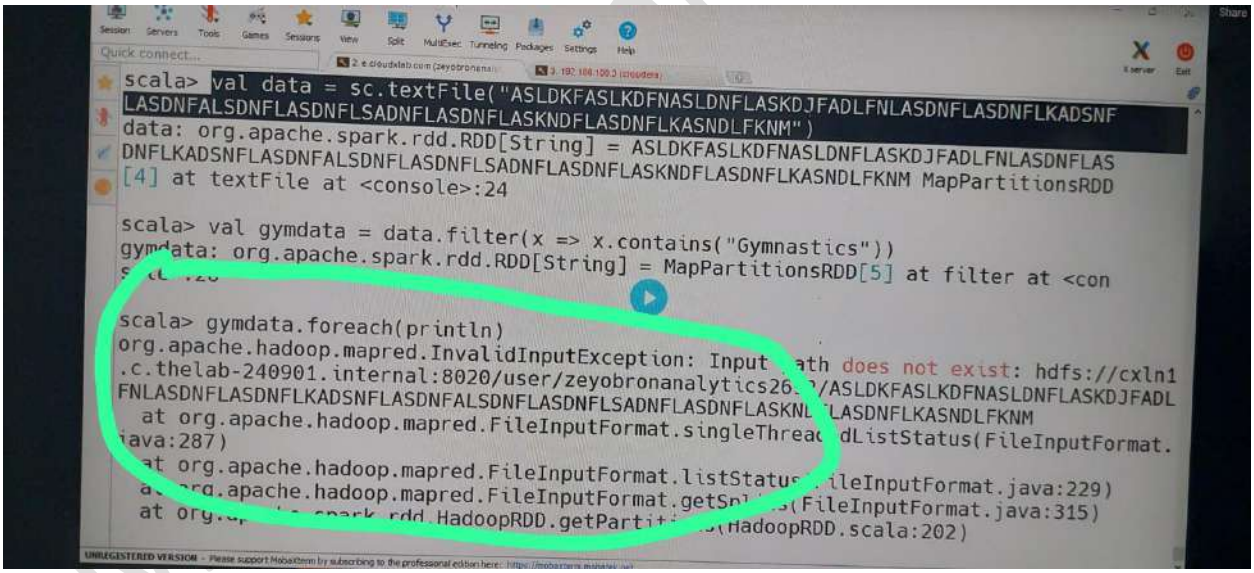
Select /*+ MAPJOIN(b) */ a.key, a.value from a join b on a.key = b.key;

Spark – Lazy Evaluation/In-memory Computation – this is the reason why Spark is faster.

Lazy evaluation Proof Execution

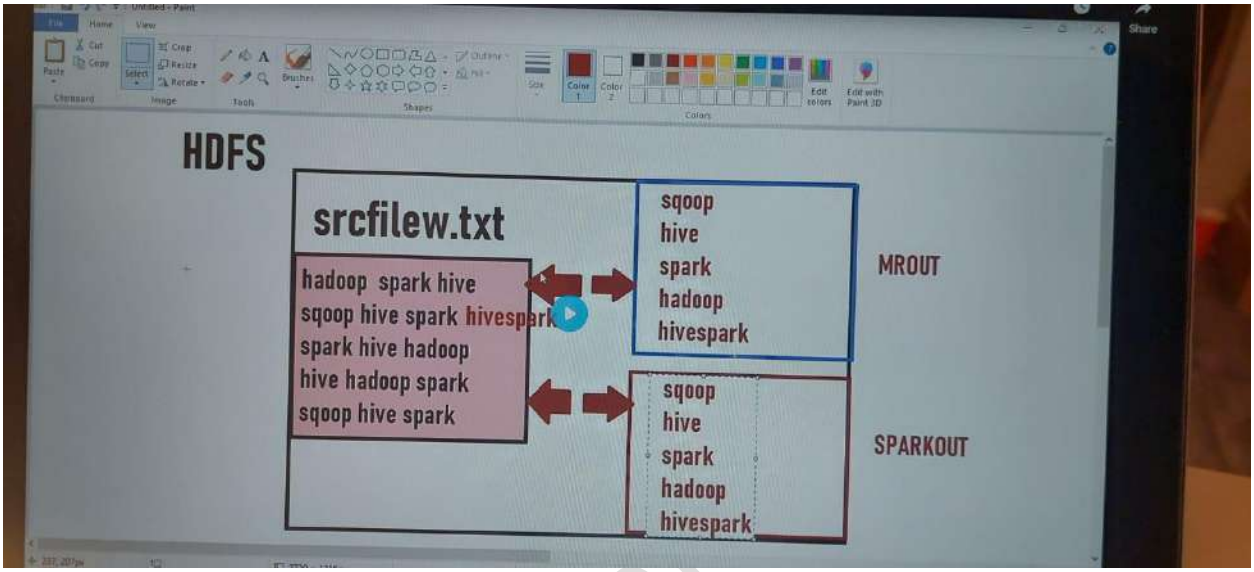
Execution:
Read the Data -- Transformation
val data = sc.testFile("data.txt")
Filter Gymnastics – Transformation
val gymdata = data.filter(x => x.contains("Gymnastics"))
Print it – Action (if we see Results)
Gymdata.foreach(println)

Proof:
we gave a wrong file path and the error is not thrown until action is executed.



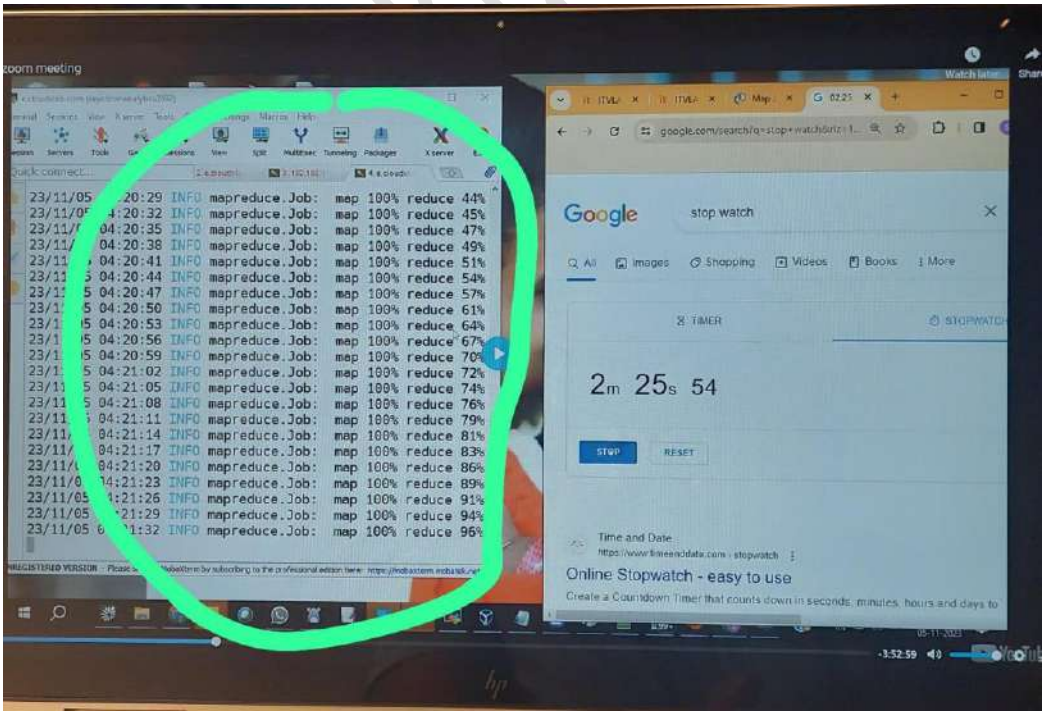
Spark vs MR Performance Timer run for Word Count

Problem Statement(checking how much time MR and spark will take):

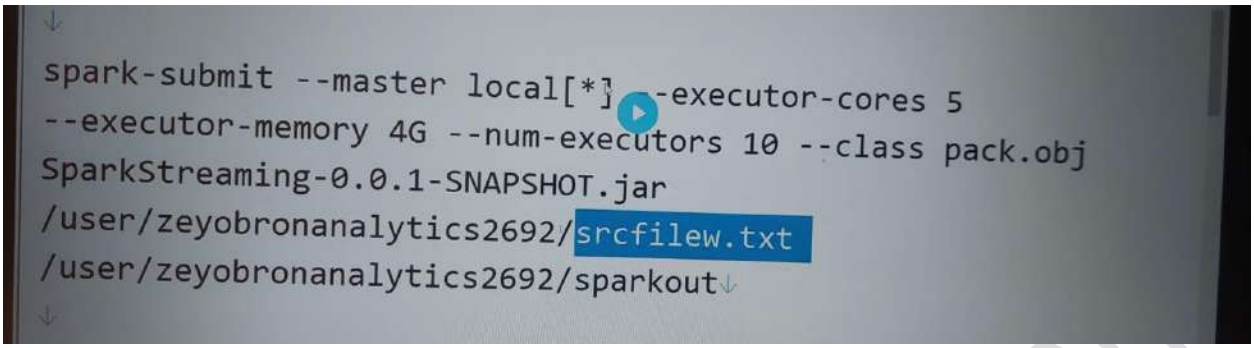


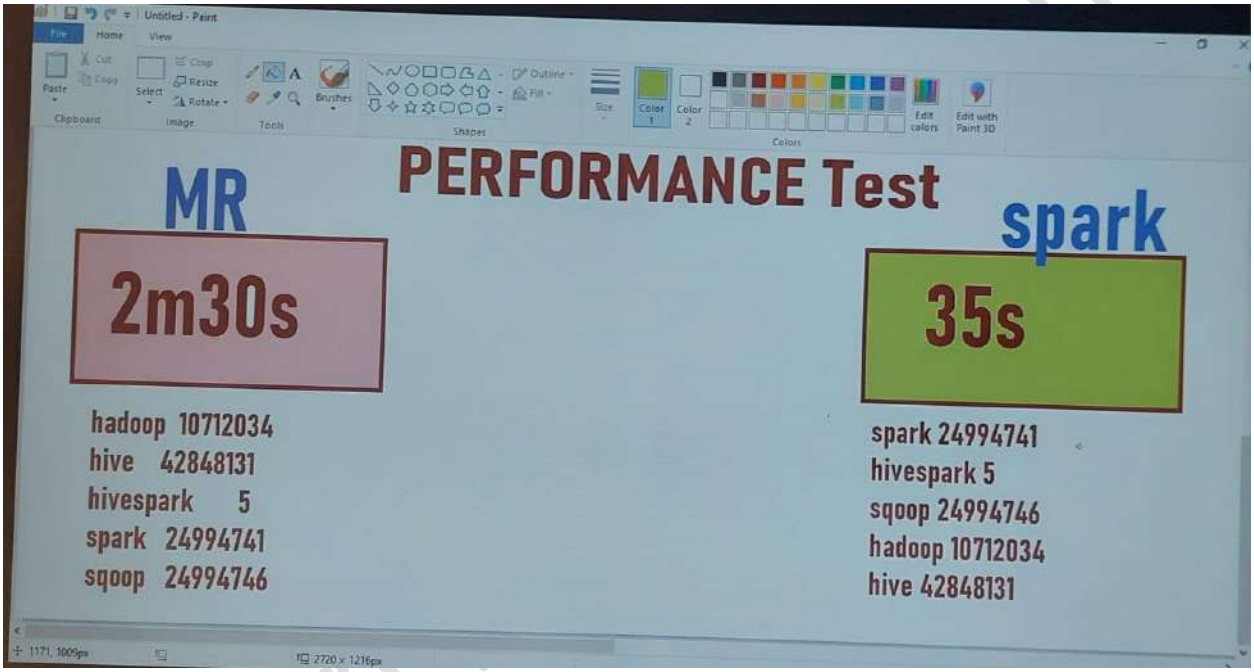
MR job:

hadoop jar mrWc-0.0.1-SNAPSHOT.jar mrWc.mrWcObj /user/zeyobronanalytics2692/srcfilew.txt /user/zeyobronanalytics2692/mrout



Spark Test:

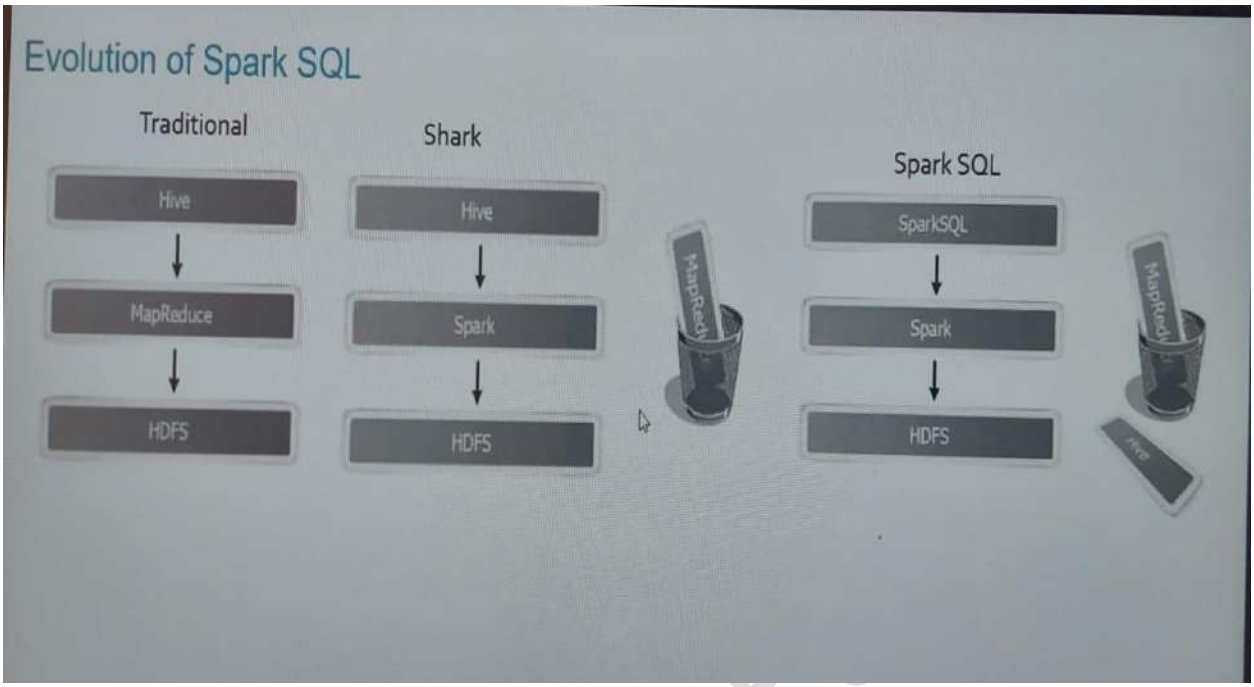




Spark Remaining Slides

Ways to process in Spark.

1) RDD (Resilient Distributed Dataset)
2) DATAFRAME (Powerful)



Spark Deployment

It means.
How Spark code is developed
How spark code is tested
How Spark code will run
How spark code is automated
How Spark code is scheduled

We have a production cluster. In that production cluster we have data in hdfs /user/zeyobronanalytics2692/data.txt. Using spark we have to read this data, filter gymnastics and write the data to gymdir.

Cluster Credentials:
hostname – e.cloudxlab.com
Username – zeyobronanalytics2692
Password – PDE304Z0

Connect to the cluster using the above details.

On Production Cluster - we will have access only to run jar. You cannot open spark-shell, you cannot run spark code(we can do this but we shouldn't)

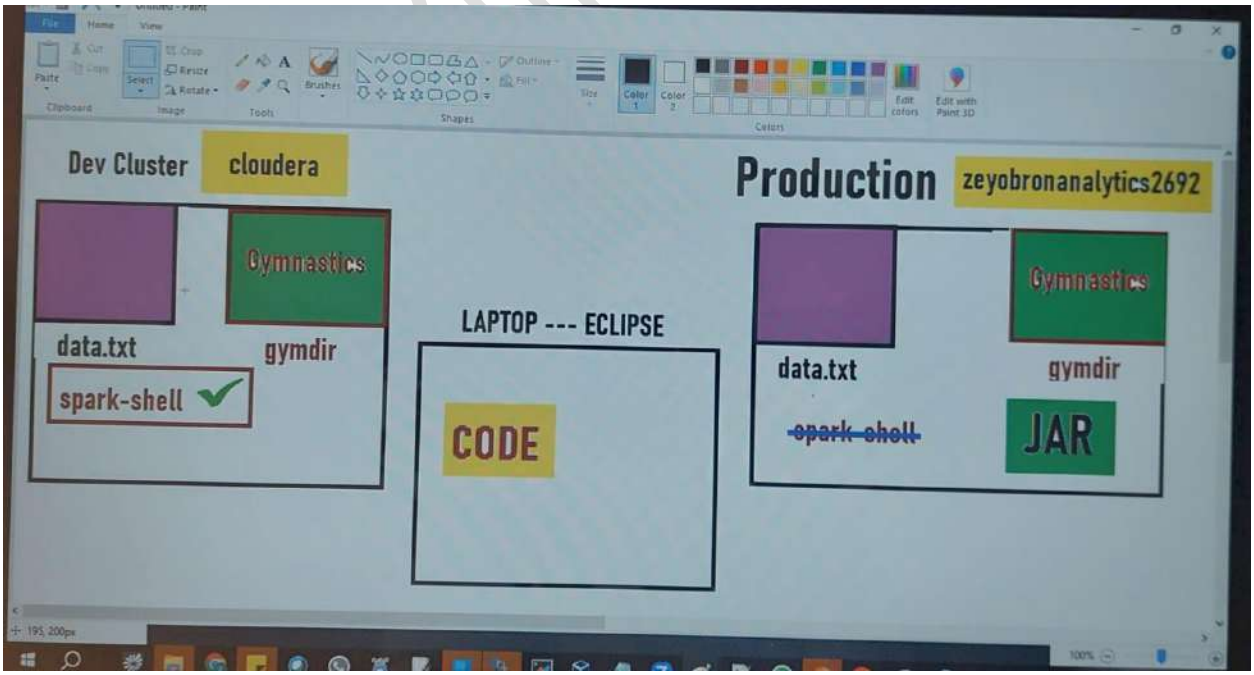
Where can I develop and test the code – to develop the code, test the results and run the code, we will have dev cluster.

Dev Cluster is Cloudera.
Hostname: <get using ifconfig>
Username: cloudera
Password: cloudera

➔ Generate Jar file in Eclipse in our local laptop

Using Winscp we can move the generated jar to the production cluster.

Run the Jar file on the production cluster.
spark-submit –class pack.obj SparkDepl-0.0.1-SNAPSHOT.jar



Spark Types of Processing Scala Intro Hands-on

Agenda Day 22

Types of Processing in Spark Deep

RDD (Resilient Distributed Datasets)– Programmatical in nature
DataFrames – SQL in nature

Sample File Read

We can do it using 2 methods.

- textFile Method from SparkContext class → spark-core.jar
- csv Method from SparkSession class → spark-sql.jar

Download Apache spark: https://spark.apache.org/downloads.html to get these Jar files

Steps:

- Create a Scala Project in your eclipse
- Add all the spark jars to that project
- Call the classes using import statements
- Initialize Object for the classes
- Access the methods to read and print it

Using SparkContext:

```
val conf = new SparkConf().setAppName("First").setMaster("local[*]");
val sc=new SparkContext(conf);
val data=sc.textFile("M:\\Learnings\\Big_Data\\DataSets\\d.txt",1)
data.foreach(println)
```

Using SparkSession:

```
val spark = SparkSession.builder().getOrCreate()
val df = spark.read.csv("M:\\Learnings\\Big_Data\\DataSets\\d.txt")
df.show()
```

Scala Declarations

val a = “Mohan”

val b =5

val lisin = List(1, 2, 3, 4)

Scala Iterations

```
val c = List("Zeyobron", 1, 2, 3)
println(c)

c.foreach(println)
```

Spark Scala Iterations List Operations

Agenda Day 23

Scala Iterations

var ls = List(5 , 6 ,7 ,8)

ls.map(x => x+2)

Lambda – terms and Conditions

1) Must mention an operation (map)
2) Left and right must be the same Variable.
3) Left Variable is an Accumulator and Right Variable is iterator.

Filter

ls.filter(x => x>2)

String Contains:

```
val c = List("Mohan","Kumar", "Mohja", "KuMohan")
println(c.filter(x => x.contains("Moh")))
```

```
val c = List("Mohan","Kumar", "Mohja", "KuMohan")
println(c.filter(x => x.equals("Mohan")))
```

```
val c = List("Mohan","Kumar", "Mohja", "KuMohan")
println(c.map(x => x.replace("Mohan","Trainer")))
```

Scala Flatmap UseCase Spark RDD
Agenda Day 24

Requirement:
Given: var list = List("A~B","C~D","E~F")
Expected: List("A","B","C","D","E","F")

Map → Do
flatMap → Do and Flatten

```
val c = List("A~B","C~D","E~F")
println(c.flatMap(x => x.split("~")))
```

Problem:
Given: List("state->TN~City->Chennai","State->UP~City->Lucknow")
Expected:
Two Lists: List("TN","UP")
List("Chennai", "Lucknow")

RDD:

The same State and City data is in a file state.txt. Now, we can read the file and get state and city separately.

```
val conf = new
SparkConf().setAppName("First").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
val sc=new SparkContext(conf);
sc.setLogLevel("ERROR")
val data = sc.textFile("M:\\Learnings\\Big_Data\\DataSets\\d.txt")
data.flatMap(x => x.split("~")).foreach(println)
```

Spark RDD File Processing usdata
Agenda Day 25

SparkConf:

```
new
SparkConf().setAppName("First").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
```

SparkContext

```
val sc=new SparkContext(conf);
sc.setLogLevel("ERROR")
val data = sc.textFile("M:\\Learnings\\Big_Data\\DataSets\\d.txt")
data.flatMap(x => x.split("~")).foreach(println)
```

SparkSession

```
val spark = SparkSession.builder().getOrCreate()
val df = spark.read.csv("M:\\Learnings\\Big_Data\\DataSets\\d.txt")
df.show()
```

take(n) – to take top n rows

```
var conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
var sparContext = new SparkContext(conf)
sparContext.setLogLevel("ERROR");
var st=sparContext.textFile("M:\\Learnings\\Big_Data\\DataSets\\usdata")
st.take(10).foreach(println)
var filtereddata = st.filter( x => x.length>200)
filtereddata.foreach(println)
```

Requirement:

- Read the data and filter rows length > 200
- Flatten the data with comma
- Remove hyphens from the result
- Concat “,zeyo” to every element
- Write the output to a file

```
var conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
    var sparContext = new SparkContext(conf)
    sparContext.setLogLevel("ERROR");
    var st=sparContext.textFile("M:\\Learnings\\Big_Data\\DataSets\\usdata")
//    st.take(10).foreach(println)
    var filtereddata = st.filter( x => x.length>200).flatMap( x =>
x.split(",")).map( x => x.replaceAll("-","")).map( x => x.concat(",zeyo"))
    filtereddata.foreach(println)


filtereddata.saveAsTextFile("M:\\Learnings\\Big_Data\\DataSets\\usdata_output
")
```

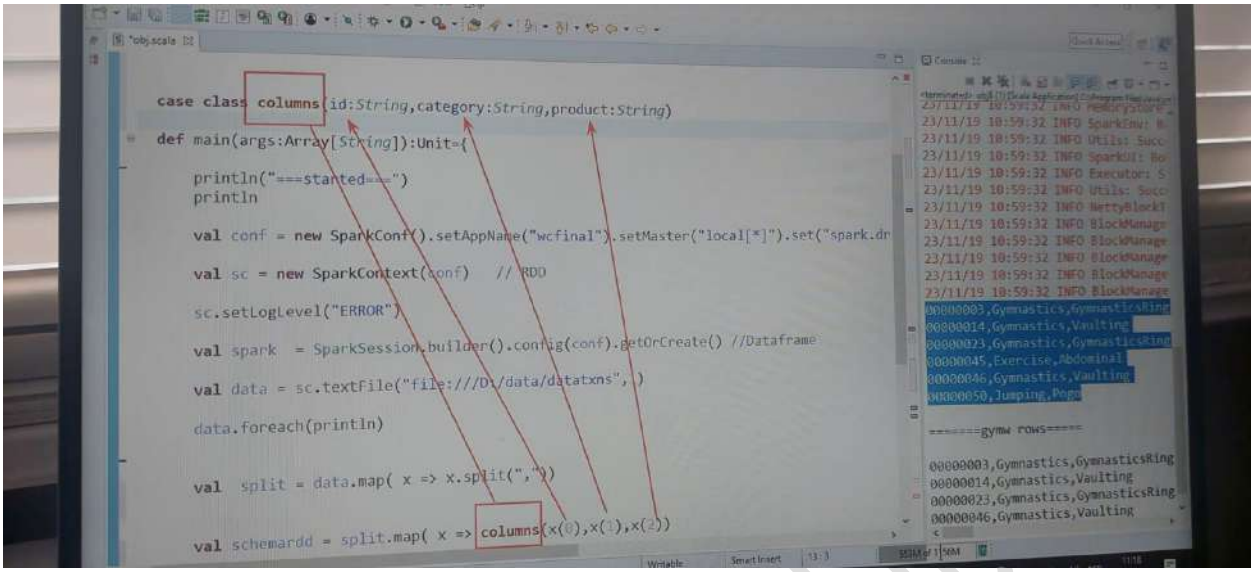
Requirement(file: usdata):

Filter rows with 3rd columns as Gymnastics and write the output as parquet

Steps:

- Separate the data with commas to control columns
- Define columns at the top
- Assign those columns to the separated splits
- Do column-based filters

```
case class columns(name:string, id:string,
product:String) → this helps to define columns in scala
object Practice2 {

  case class columns(id:String, category:String, productType:String);

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

    var conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
    var sparContext = new SparkContext(conf)
    sparContext.setLogLevel("ERROR");
    var
st=sparContext.textFile("M:\\Learnings\\Big_Data\\DataSets\\datatxns")
    st.map(x => x.split(",")).map(x => columns(x(0),x(1),x(2)))
      .filter(x => x.productType.contains("Gymnastics")).foreach(println)
```



Tasks:

1) Schema Rdd filter Task above

2)What is DAG in Spark

3) Read Us Data, Filter rows contains “LA”, flatten the data with comma

Spark DataFrame Conversion rdd dataframe reads

Agenda Day 26

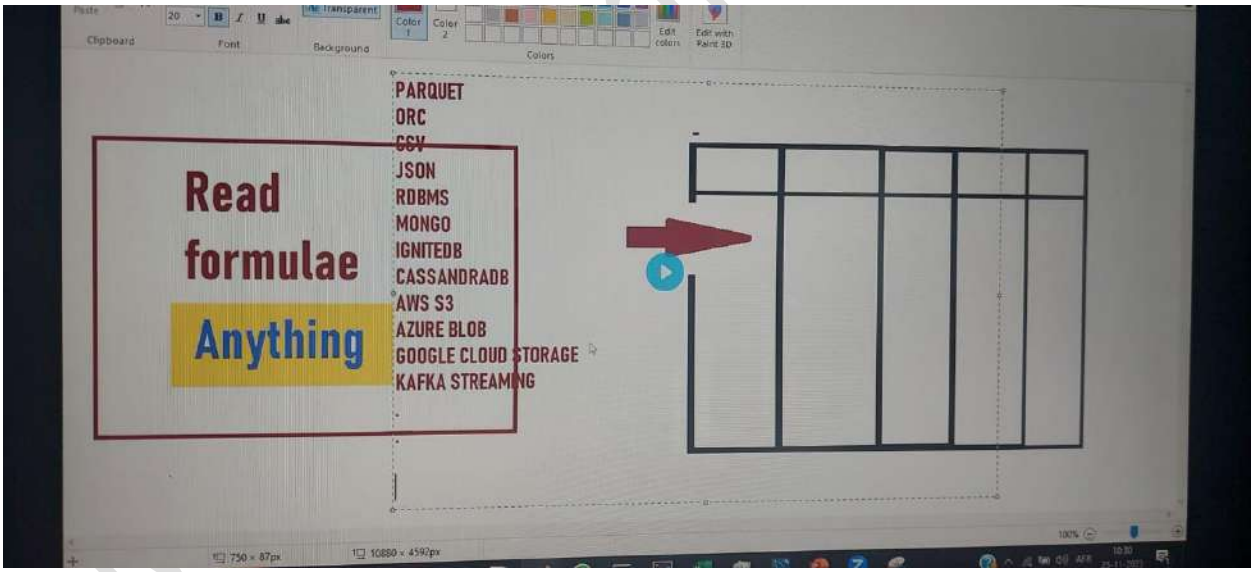
Spark dataframe Conversion

- ➔ Parquet writes using spark rdd is impossible. To write it as Parquet, we need dataframe in place. To do this, Convert that filtered Schema rdd to Dataframe.
- ➔ To convert filtered schema rdd (st in the above code) to dataframe, we need to use “.toDF()”. df.write.parquet(“file:///<filepath>”)

➔To convert df to rdd, we need to use df.rdd

```scala
import org.apache.spark.sql.SparkSession
import org.apache.spark.{SparkConf, SparkContext}

object Practice2 {

  case class columns(id:String, category:String, productType:String);

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

    val conf = new SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostname","localhost")
    val sparContext = new SparkContext(conf)
    sparContext.setLogLevel("ERROR");

    val spark = SparkSession.builder().config(conf).getOrCreate()
    import  spark.implicits._

    val st=sparContext.textFile( path = "M:\\Learnings\\Big_Data\\DataSets\\datatxns")
    val schemaRDD=st.map(x => x.split( regex = ",")).map(x => columns(x(0),x(1),x(2)))
      .filter(x => x.productType.contains("Gymnastics"))
    schemaRDD.foreach(println)
    val df = schemaRDD.toDF()
    df.show()

  }
```

Spark dataframe creation



Read Formulae: spark read format options load.

Reading devices.json as a dataframe:

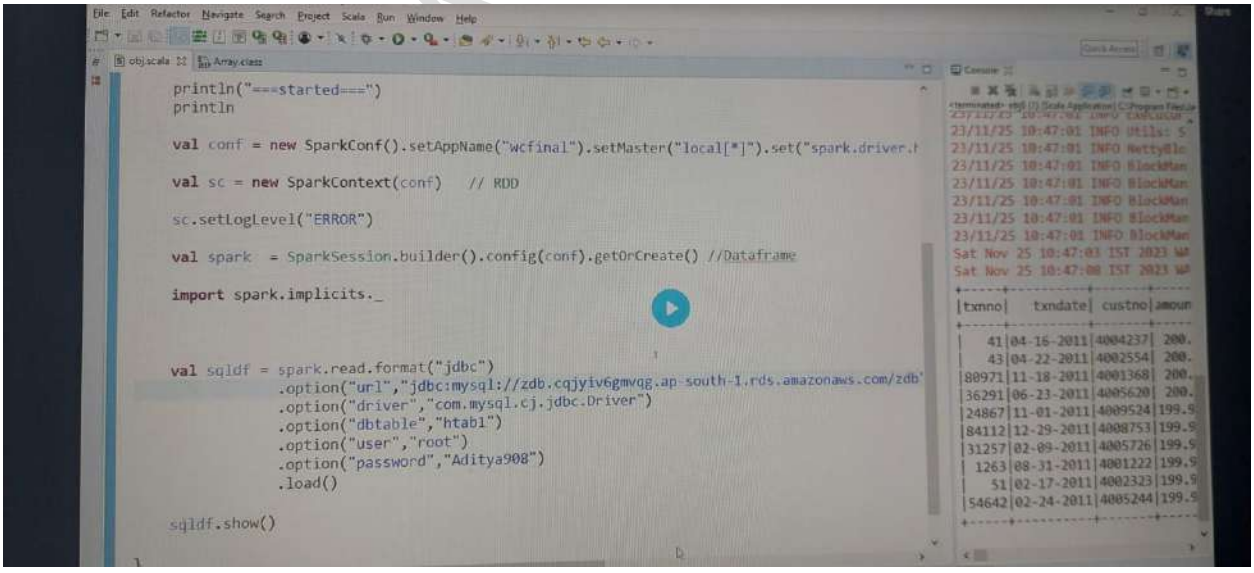
```
val note =
spark.read.format("json").load("M:\\Learnings\\Big_Data\\DataSets\\devices.json")
note.show()
```

```
Reading orcfile.orc (Serialized data)as a dataframe:
val conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
val sparContext = new SparkContext(conf)
sparContext.setLogLevel("ERROR");

val spark = SparkSession.builder().config(conf).getOrCreate()
import  spark.implicits._

val note =
spark.read.format("orc").load("M:\\Learnings\\Big_Data\\DataSets\\orcfile.orc
")
note.show()
```

Reading parfile.parquet as a dataframe:

```
val note =
spark.read.format("parquet").load("M:\\Learnings\\Big_Data\\DataSets\\parfile
.parquet")
note.show()
```

Reading from SQL DataBase:



Spark file reads formulae

Requirement : Read devices.json and filter records where latitudes>40

Solution Steps: Assign name and Shoot SQL queries on the name.

```
val conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
val sparContext = new SparkContext(conf)
sparContext.setLogLevel("ERROR");

val spark = SparkSession.builder().config(conf).getOrCreate()
import  spark.implicits._

val note =
spark.read.format("json").load("M:\\Learnings\\Big_Data\\DataSets\\devices.js
on")
note.show()

note.createOrReplaceTempView("animal")
val finaldf = spark.sql("select * from animal where lat>40")
finaldf.show();
```

Requirement:

If we want to make first row as header:
spark.read.format(“csv”).option(“header”,”true”).load(“<filepath>”)

```

val csvdf =
spark.read.format("csv").option("header","true").load("M:\\Learnings\\Big_Dat
a\\DataSets\\df.csv")
csvdf.show()
```

Match the datatypes in Schema to data

```
case class columns(id:Int, category:String, productType:String);

val schemaRDD=st.map(x => x.split(",")).map(x =>
columns(x(0).toInt,x(1),x(2)))
```

Spark Dataframe writes avro reads

Agenda 27

AVRO/XML Reads

Spark by default supports only csv, parquet, orc, json.
To get Spark Environment we added 224 Jars, these Jars can support only 4 file formats. If we need extra like avro,xml, cloud read, rdbms, nosql, streaming platforms..these jars are not enough.
If we need others(avro, xml..etc) to work we need Extra Jars.

Currently we are using Spark – 2.4.7 and Scala – 2.11 versions.

Get the Jars Required for AVRO

Home » org.apache.spark » spark-avro_2.12 » 2.4.7

Spark Avro » 2.4.7

Spark Avro

License	Apache 2.0
Tags	serialization avro spark apache protocol
HomePage	http://spark.apache.org/
Date	Sep 12, 2020
Files	jar (104 KB) View All
Repositories	Central
Ranking	#3574 in MvnRepository (See Top Artifacts)
Used By	121 artifacts
Scala Target	Scala 2.12 (View all targets)
Vulnerabilities	Vulnerabilities from dependencies: CVE-2023-22946 CVE-2020-15250

```
val csvdf =
spark.read.format("avro").load("M:\\Learnings\\Big_Data\\DataSets\\part.avro"
)
    csvdf.show()
```

Creating a Maven Project:

Task: Difference between jar and FAT jar

Try this is IntelliJ

Dataframe writes

Requirement:

Read devices.json, Filter humidity>50, write it as csv file with headers

Write Formulae: fianldf write format options mode save

Procdf.write.format(“csv”).options(“header”,”true”).mode(“overwrite”).save(“file:///<filepath_to_save>”)

For the Write to work, we need to install winutils:

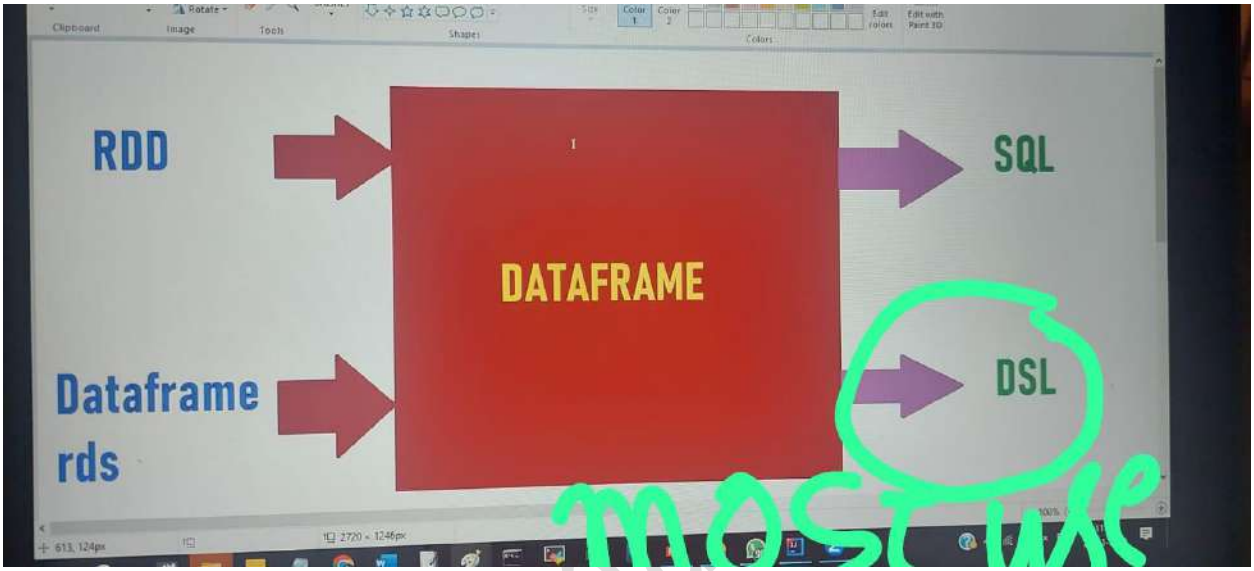
Create a folder named as hadoop/bin in any of the directories and place winutils.exe file in that folder. In my system it’s at M:\Learnings\Spark_Learning\hadoop-3.3.5\bin

and in our code, we have to set a property as System.setProperty(“haddop.home.dir”,” M:\Learnings\Spark_Learning\hadoop-3.3.5”)

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

  System.setProperty("hadoop.home.dir",
"M:\\Learnings\\Spark_Learning\\hadoop-3.3.5")
  val conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostn
ame","localhost")
  val sparContext = new SparkContext(conf)
  sparContext.setLogLevel("ERROR");

  val spark = SparkSession.builder().config(conf).getOrCreate()
  import  spark.implicits._

  val csvdf =
```

```
spark.read.format("avro").load("M:\\Learnings\\Big_Data\\DataSets\\part.avro"
)

csvdf.write.format("csv").option("header","true").mode("overwrite").save("M:\
\Learnings\\Big_Data\\DataSets\\csvout")
```

Spark DSL




```
val note =
spark.read.format("json").load("M:\\Learnings\\Big_Data\\DataSets\\devices.js
on")
note.filter("humidity > 95").show()
```

Tasks:

1) Read orcfile.orc, Filter age > 10 using sparksql, show the dataframe
2) Difference between jar and fatjar
3) complete SQL Document

Spark dsl deep dive select drop filters

Agenda 28

Providing Columns to our day can be done in two ways:

1)Not an optimal way (problem if we have more columns – we cannot hardcode all of them):
spark.read.format(“csv”).load(“file:///D:/data/dt.txt”).toDF(“id”,”tdate”,”amount”,”category”)

2) other way: Struct Type – this can be dynamically generated with the values in a file

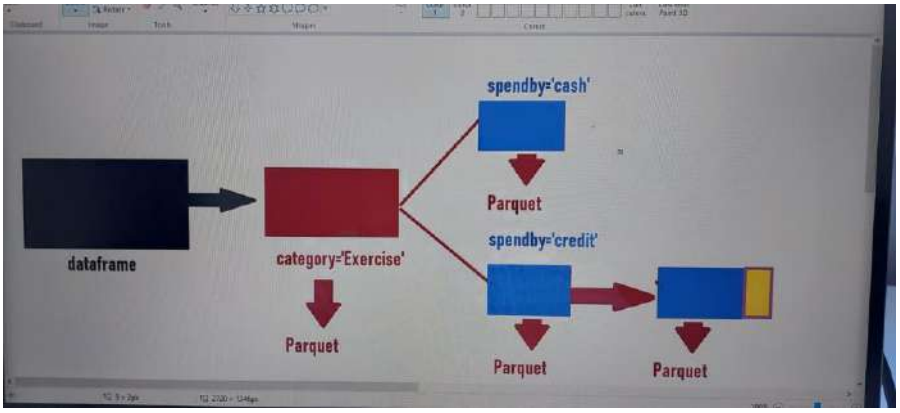
import: import org.apache.spark.sql.types._

val structschema = StructType(Array(StructField(“id”,StringType,true), StructField(“tdate”,StringType,true)))

spark.read.format(“csv”).schema(structschema).load(“file:///D:/data/dt.txt”)

Spark DSL(Domain Specific Language)

Requirement:
read dt.txt and assign columns
From the results filter category=’Excercise’ Write it as parquet
From the results filter spendby=’cash’ Write it as parquet
From the results filter spendby=’credit’ Write it as parquet
From the credit filter – attach one column – name – zeyo write it as parquet



1st way(sql): Creating multiple temp views and applying SQL query on that temp view

```
val note =
spark.read.format("json").load("M:\\Learnings\\Big_Data\\DataSets\\devices.js
on")
note.show()

note.createOrReplaceTempView("animal")
val finaldf = spark.sql("select * from animal where lat>40")
finaldf.show()

finaldf.createOrReplaceTempView("latitudf")
val furtherdf= spark.sql("select * from latitudf where lat>90")
furtherdf.show()
```

2nd way (DSL): withcolumn is to append a column to df

```
val spark = SparkSession.builder().config(conf).getOrCreate()
import  spark.implicits._

val note =
spark.read.format("json").load("M:\\Learnings\\Big_Data\\DataSets\\devices.js
on")
val mediumhumdf= note.filter("humidity > 40")
mediumhumdf.show()

val highhumdf = mediumhumdf.filter("humidity > 90")
highhumdf.withColumn("name",lit("Mohan")).show()
```

Spark Filters in DSL

Select and Drop

```
System.setProperty("hadoop.home.dir", "M:\\Learnings\\Spark_Learning\\hadoop-3.3.5")
val conf = new SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark.driver.hostname", "localhost")
val sparContext = new SparkContext(conf)
sparContext.setLogLevel("ERROR");

val spark = SparkSession.builder().config(conf).getOrCreate()
import spark.implicits._

val note = spark.read.format("csv").option("header","true").load("M:\\Learnings\\Big_Data\\DataSets\\dtnew.txt")
note.show()

//select Specific Column from table
val seldf = note.select("id", "category")
seldf.show()

//drop a specific column
//Drop operations will not effect the original data, these are just session based
//spark will
val dropdf = note.drop("product")
dropdf.show()
```

Single Column Filter

```
//Single column Filters using DSL Category = "Excercise"
//During DSL Operations we have to use col operations
val filcat = note.filter(col("category")==="Exercise")
filcat.show()
```

Multiple Column Filter

And operator

```
//Multiple column Filter
//category = Excercise and spendby = "cash"
val MulFilter = note.filter(col("category") === "Exercise" && col("spendby") === "cash")
MulFilter.show()
```

Or operator

```
//Multiple column Filter
//category = Excercise and spendby = "cash"
val MulORFilter = note.filter(col("category") === "Exercise" ||
col("spendby") === "cash")
MulORFilter.show()
```

Multiple Value Filter to the same column (isin)

```
//Multiple Value filter - Category = "Excercise" or spendby = "cash"
val MulvalFilter = note.filter(col("category") isin ("Exercise","Team
Sports"))
MulvalFilter.show()
```

Contains (like)

```
//Contains
val likefil = note.filter(col("product") like
("%Gymnastics%"))
likefil.show()
```

Spark DSL Filters Expression Intro
Agenda 29

isNull

```
//is null
val filNull = note.filter(col("product") isNull)
filNull.show()

val filSecWayNull = note.filter(col("product").isNull)
filSecWayNull.show()
```

isNotNull

```
//isNotNull
val filNotNull = note.filter(col("product") isNotNull)
```

```
filNotNull.show()

val filSecWayNotNull =
note.filter(col("product").isNotNull)
filSecWayNotNull.show()
```

Not

```
//not
val filnot = note.filter(!(col("category") ===
"Exercise"))
filnot.show()
```

```
val MulNotFilter = note.filter(!(col("category") ===
"Exercise" && col("spendby") === "cash"))
MulNotFilter.show()
```

Expressions

```
val selExpre = note.selectExpr("id", "upper(category)")
selExpre.show()
```

Task:

1) Read dtnew.txt, Filter SpendBy do not contain Cash
   Create a new temp view, Attach a column as status with value as 'Zeyo'
2) Convert Uber Data – Date to day

Spark dsl Expressions with Column joins intro

Agenda Day 30

Spark Expressions

Alias

```
val selExpre = note.selectExpr("id", "upper(category)
as category")
selExpre.show()
```

case

```
val caseExp = note.selectExpr(
  "id",
  "tdate",
  "amount",
  "upper(category) as category",
  "product",
  "spendby",
  "case when spendby = 'cash' then 0 else 1 end as
status"
)
caseExp.show()
```

Spark Conditions

To process/ change two columns we are selecting all columns
We need with Column to solve this, with column – with pick the required column process and places back at its position.

withColumn(Column, Expression)

```
val withdf = note.withColumn("category",
expr("upper(category)"))
withdf.show()
```

if exists column is given it will impact, if new column is given it will add at last

```
val condwithdf = note.withColumn("status", expr("case
when spendby = 'cash' then 0 else 1 end"))
condwithdf.show()
```

Spark Joins

Inner Join

```
//joins
val cust = spark.read.format("csv").option("header",
"true").load("M:\\Learnings\\Big_Data\\DataSets\\cust.c
sv")
```

```
cust.show()

val prod = spark.read.format("csv").option("header",
"true").load("M:\\Learnings\\Big_Data\\DataSets\\prod.c
sv")
prod.show()

//inner join
val inner = cust.join(prod, Seq("id"), "inner")
inner.show()
```

Left join

```
val left = cust.join(prod, Seq("id"), "left")
left.show()
```

Right Join

```
//right join
val right = cust.join(prod, Seq("id"), "right")
right.show()
```

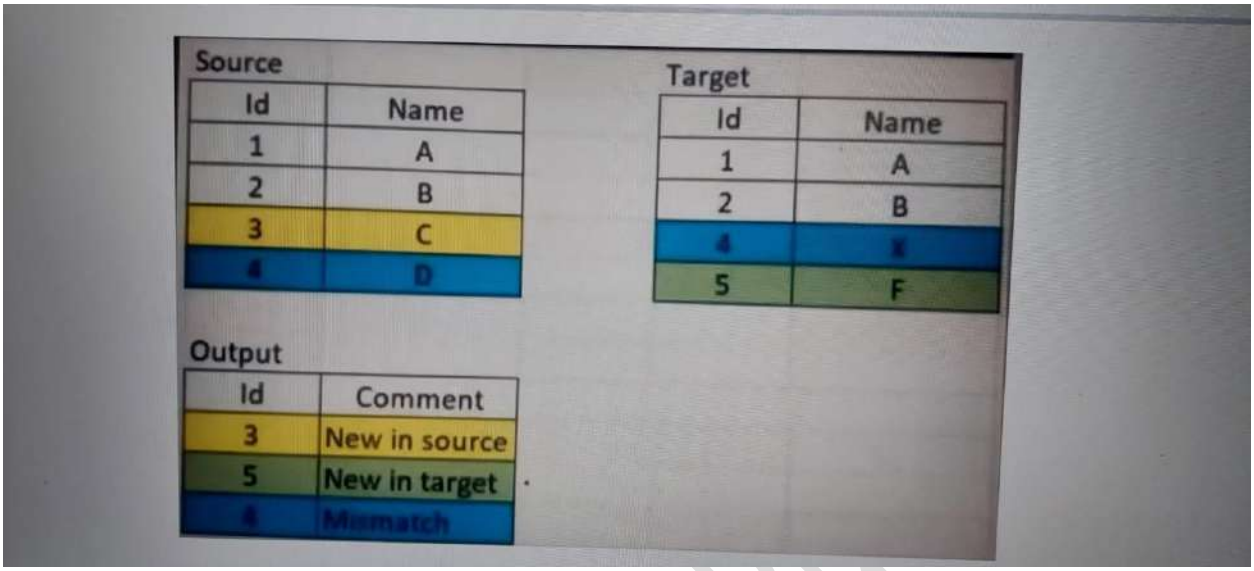
Full join

```
//full join
val full = cust.join(prod, Seq("id"), "full")
full.show()
```

OrderBy

```
//full join
val fullOrderBy = cust.join(prod, Seq("id"),
"full").orderBy(col("id"))
fullOrderBy.show()
```

Spark Scenario Solving



Source

Id	Name
1	A
2	B
3	C
4	D

Target

Id	Name
1	A
2	B
4	X
5	F

Output

Id	Comment
3	New in source
5	New in target
4	Mismatch

```
//Scenario
val source = spark.read.format("csv").option("header",
"true").load("M:\\Learnings\\Big_Data\\DataSets\\source
.csv")
source.show()

val target = spark.read.format("csv").option("header",
"true").load("M:\\Learnings\\Big_Data\\DataSets\\target
.csv")
target.show()

source.join(target, Seq("id"), "full")
  .withColumn("comment", expr("case when sname = tname
then 'Match' else 'Mismatch' end"))
  .filter( ! (col("comment") === "Match"))
  .withColumn("comment", expr("case when tname is null
then 'New in Target' when sname is null then 'New in
Source' else comment end"))
  .show()
```

Spark Joins Anti Scenarios AWS Integration.
Agenda Day 31

Monotonically Increasing id option in with column:

```
val source = spark.read.format("csv").option("header",
"true").load("M:\\Learnings\\Big_Data\\DataSets\\source
.csv")
source.withColumn("cid",
monotonically_increasing_id()).show();
```

```
+---+-----+---+
| id|sname|cid|
+---+-----+---+
|  1|    A|  0|
|  2|    B|  1|
|  3|    C|  2|
|  4|    D|  3|
+---+-----+---+
```

Scenario:

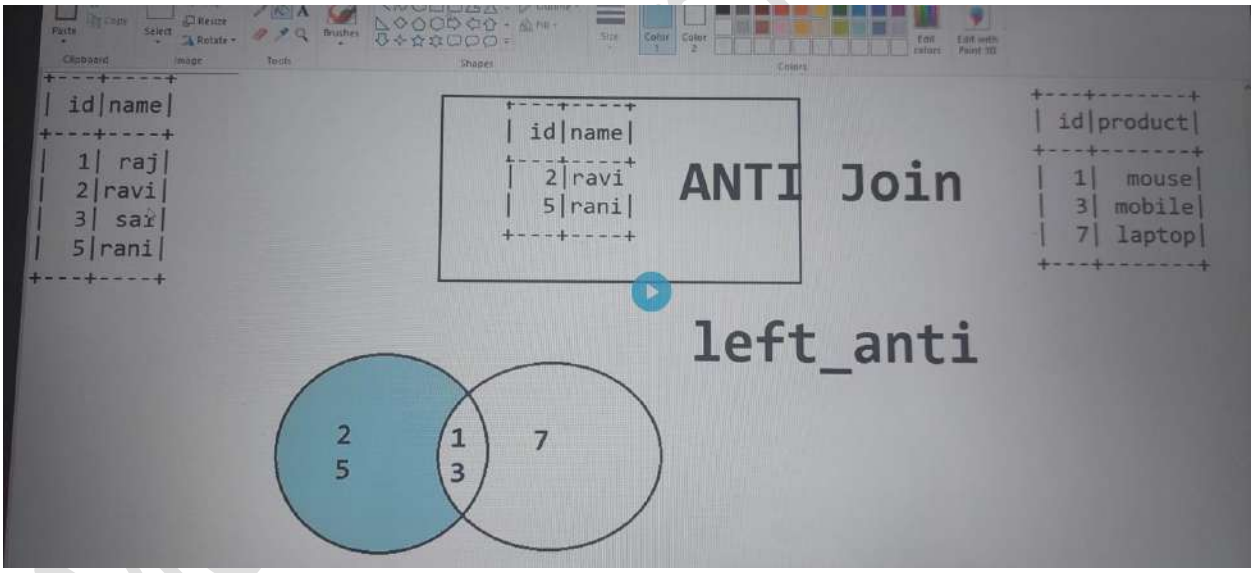
Given:
val List1 = List (“1”,”2”,”3”)
val List2 = List (“One”,”Two”, ”Three”)

Output:
in a df column, we should get
1 is One
2 is Two
3 is Three

```
System.setProperty("hadoop.home.dir",
"M:\\Learnings\\Spark_Learning\\hadoop-3.3.5")
val conf = new
SparkConf().setAppName("Mohan").setMaster("local[*]").set("spark
```

```
.driver.hostname", "localhost")
val sparContext = new SparkContext(conf)
sparContext.setLogLevel("ERROR");

val spark = SparkSession.builder().config(conf).getOrCreate()
import spark.implicits._

val list1= List("1", "2", "3");
val list2 = List("One", "Two", "Three");

val listdf1 = list1.toDF("nid").withColumn("cid",
monotonically_increasing_id());
val listdf2 = list2.toDF("nname").withColumn("cid",
monotonically_increasing_id());
listdf1.join(listdf2, Seq("cid")).withColumn("value",
expr("concat(nid, ' is ', nname)")).show()
```

ANTI joins:



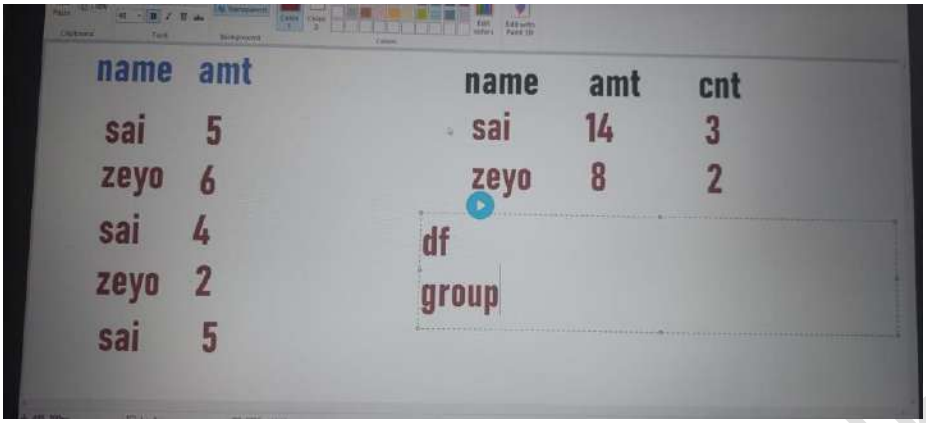

```
val anti = source.join(target, Seq("id"), "left_anti");
anti.show()
```

No, Right_anti for reverse, to get 7, we just have to suffle tables

```
val anti = target.join(source, Seq("id"), "left_anti");
anti.show()
```

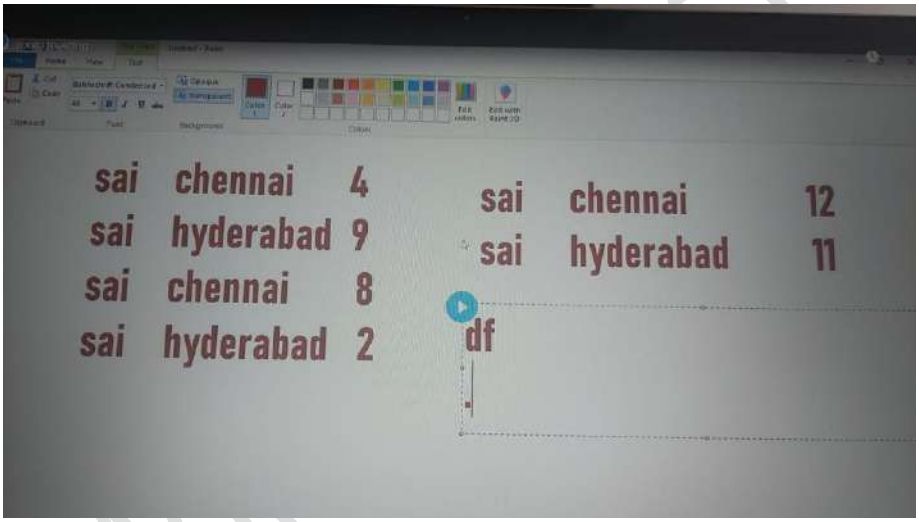
Aggregations:

File: agg.csv



df.groupBy("name").agg(sum("amt").cast(IntegerType).as("Total"))

df.groupBy("name").agg(sum("amt").cast(IntegerType).as("Total"), count("amt").as("cnt"))

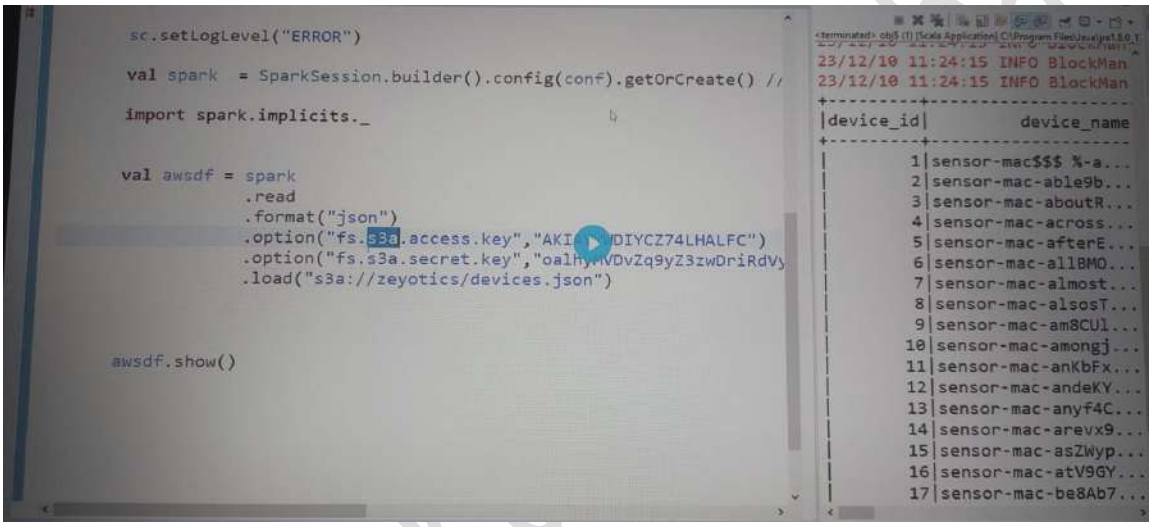


df.groupBy("name","loc").agg(sum("amt"))

AWS Integration – Reading the data from AWS S3 bucket.

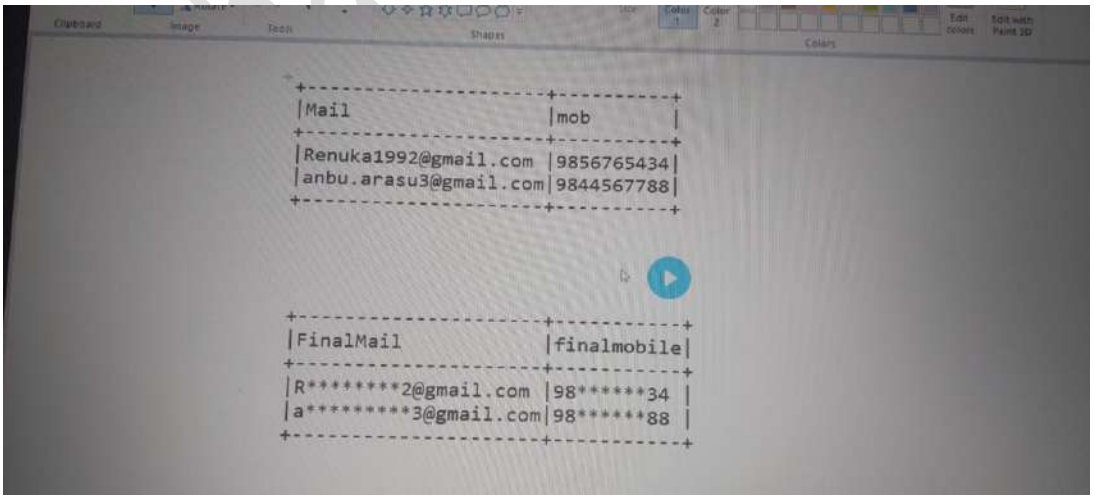
we need these details, Bucket- object name, access key, and secret key.

```
val awsdf = spark.read.format(“json”)
                .option(“fs.s3a.access.key”,”<Access-Key>”)
                .option(“fs.s3a.secret.key”,”<Secret_key>”)
                    .load(“<bucket_Object_Path>”)
```



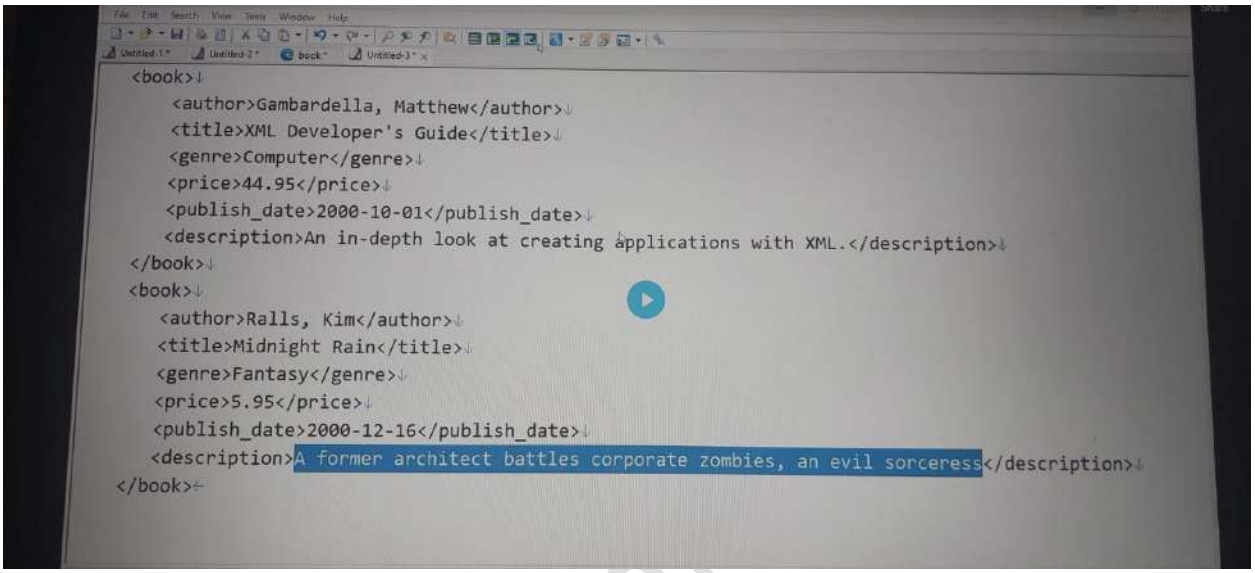
Task :

mail.csv

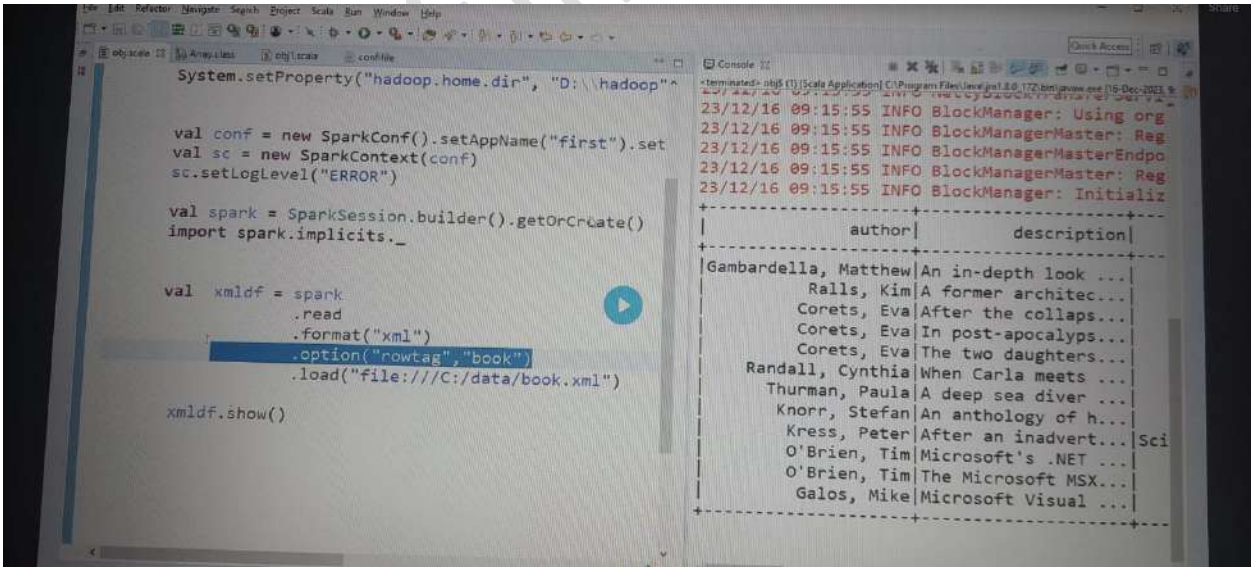


Spark Partition By XML modes Complex data intro scenario Agenda Day 32

Converting XML to dataFrame

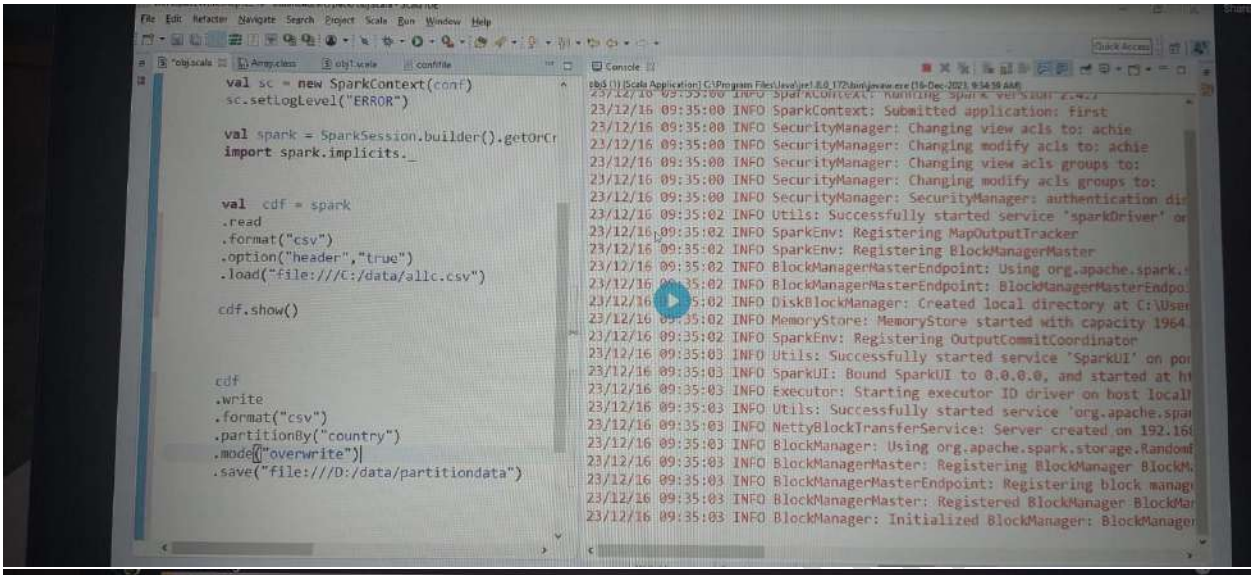


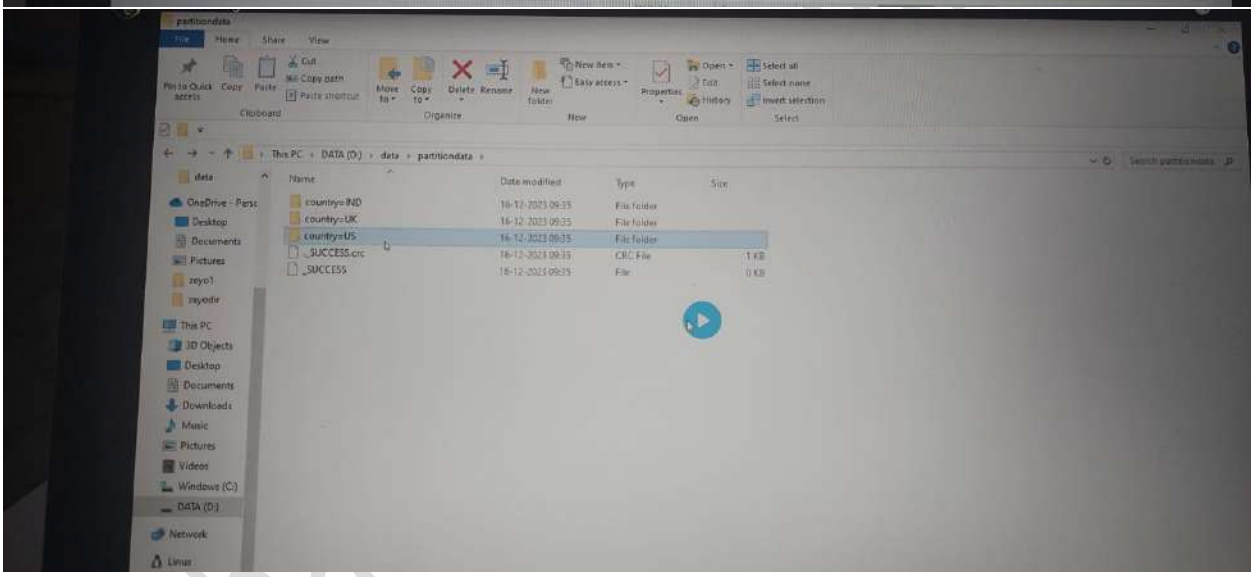
Download Spark-Xml 2.11 Jar and add the jar. We also need to give the start and end tag.




```
val xmread = spark.read.format("xml").option("rowtag",
"book").load("M:\\Learnings\\Big_Data\\DataSets\\book.xml")
xmread.show()
```

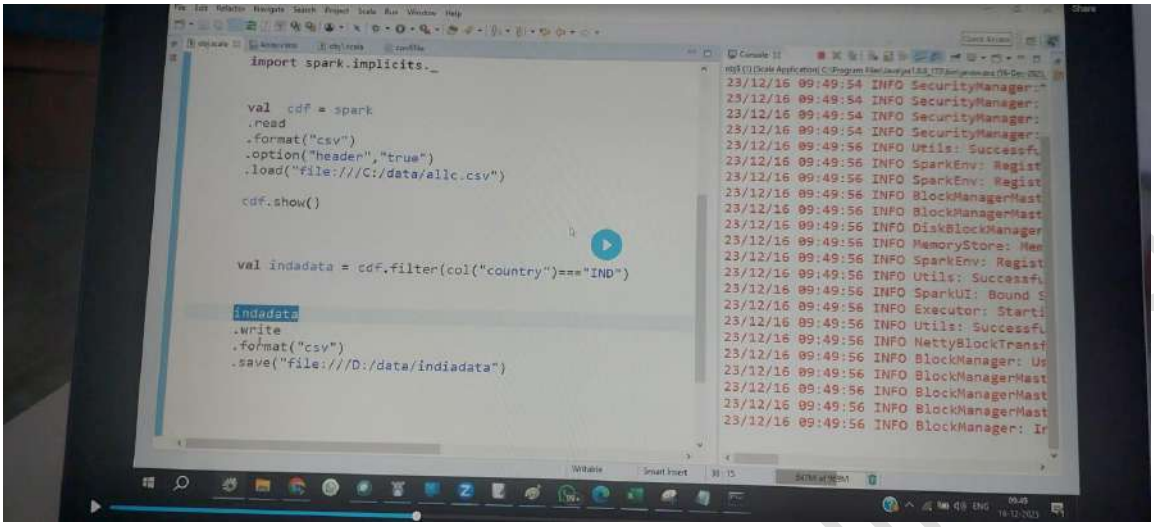
PartitionBy



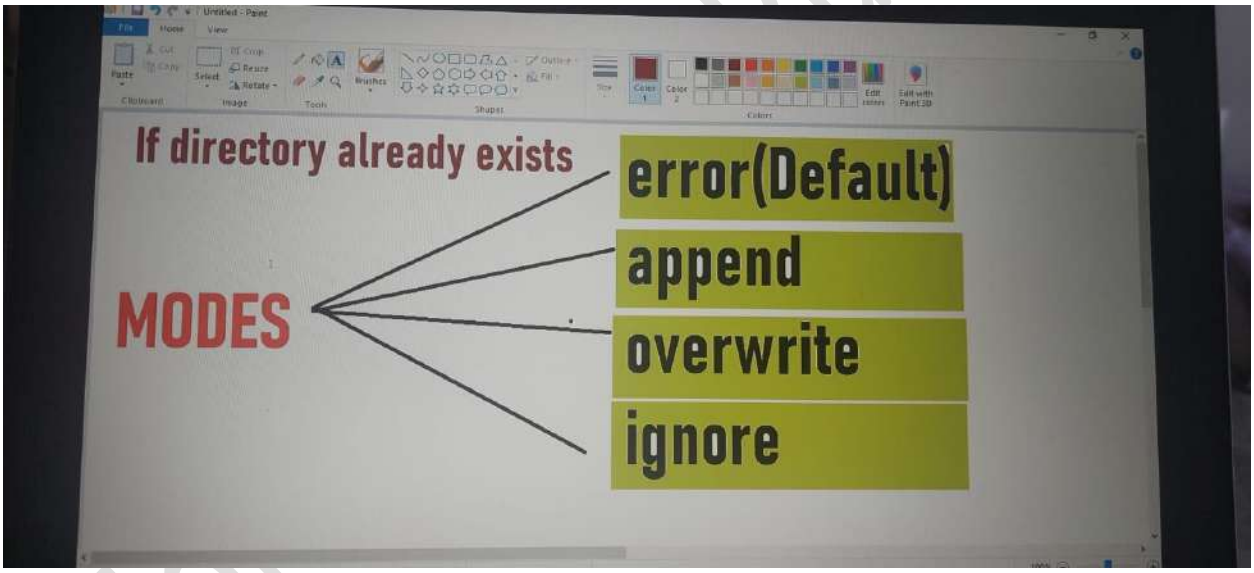



```
val xmread =
spark.read.format("csv").option("header","true").load("M:\\Learn
ings\\Big_Data\\DataSets\\allc.csv")
xmread.write.format("csv").partitionBy("country","chk").mode("ov
erwrite").save("M:\\Learnings\\Big_Data\\DataSets\\partitiondata
")
```

Save Modes

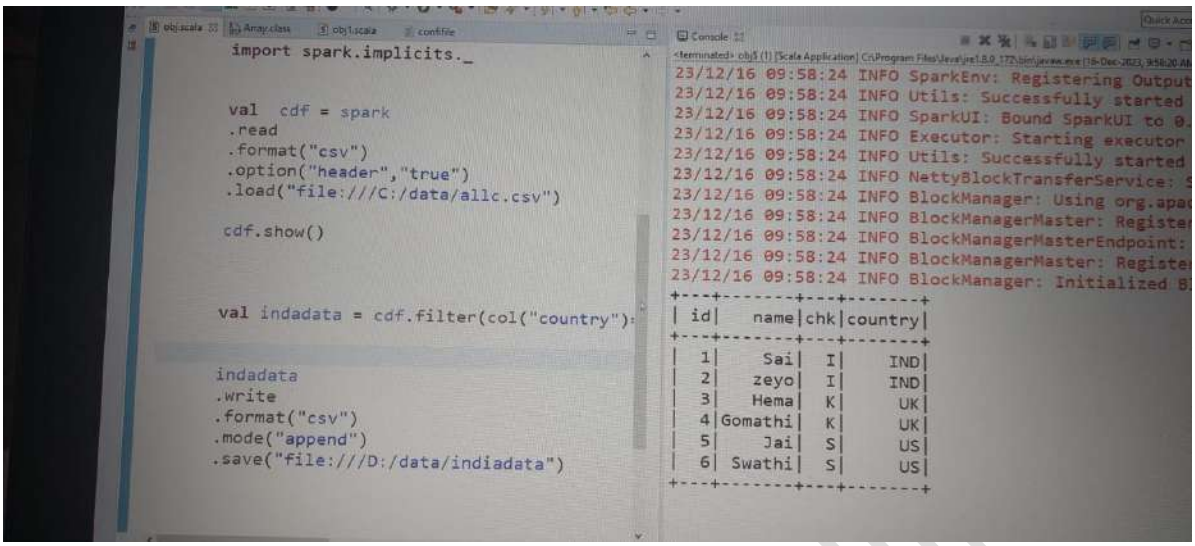


When we save, Spark will always expect non-existing directories



Error: throws out an error when the folder exists
Append: Appends a part file in the same directory
overwrite: overwrites the existing part file
Ignore: no error will be thrown, Just ignore the write

Code to pass the save mode



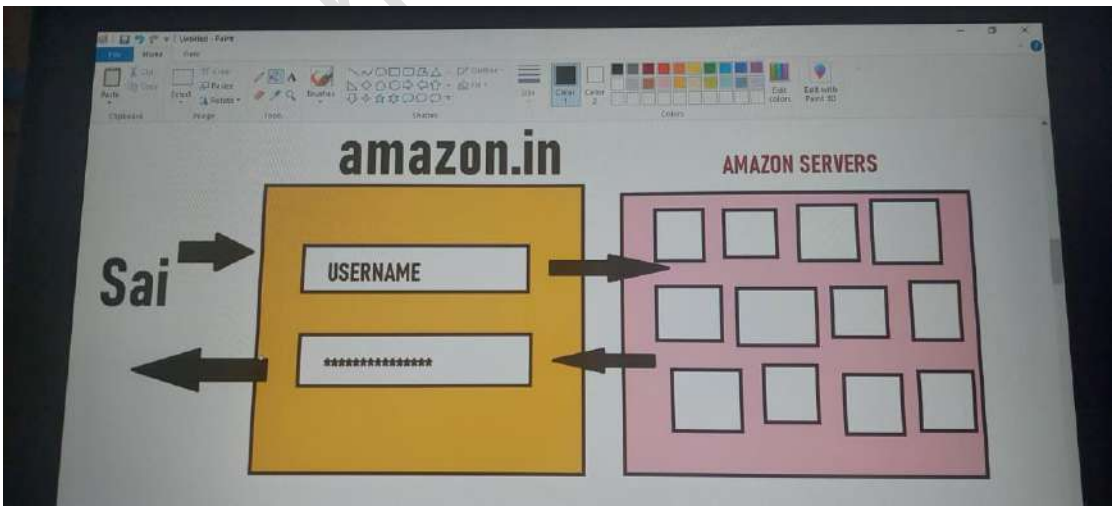
Complex data Intro

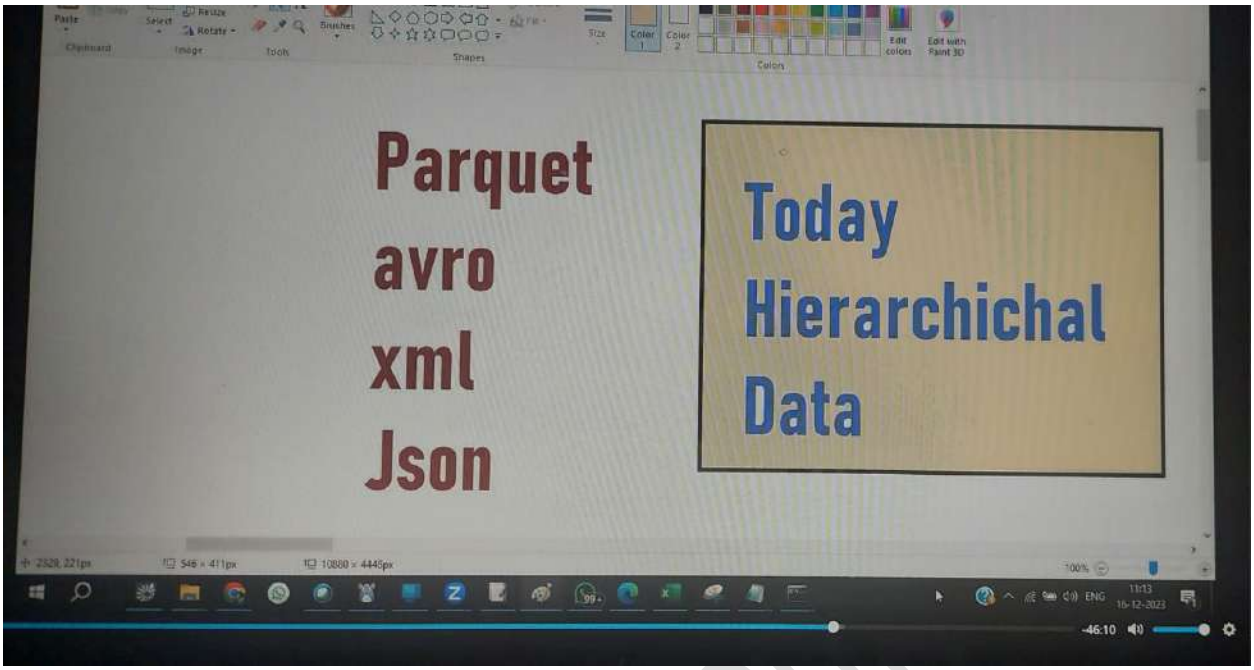
cdf.printSchema() – to print the schema

datafile: transactions.xml

Complex data will have so many hierarchies – more Nested Data

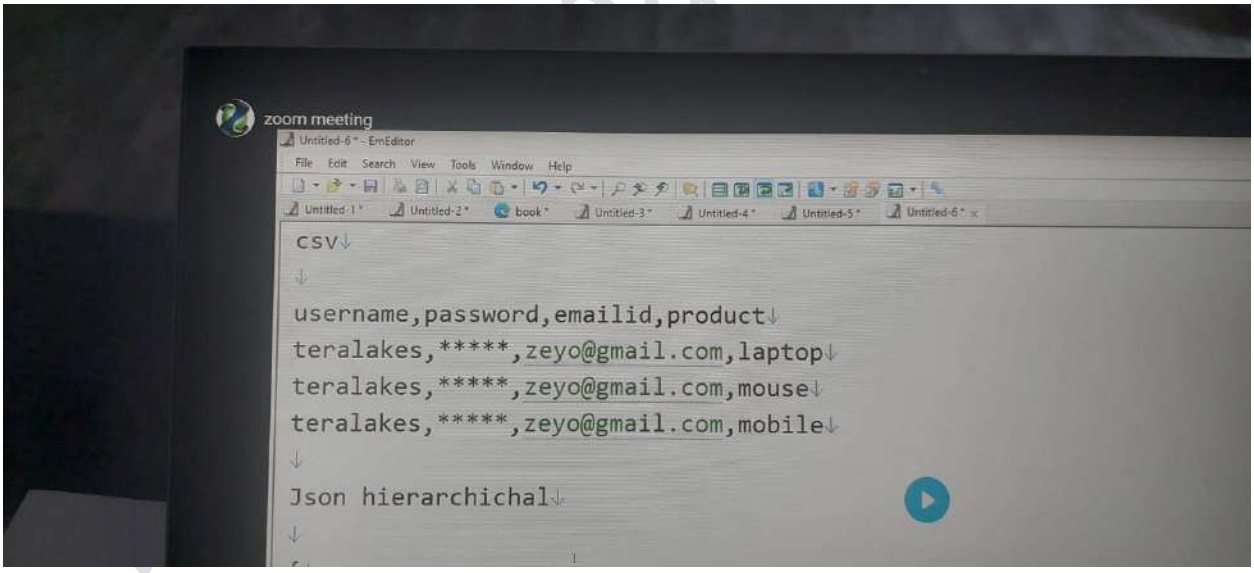
Most of the Hierarchical data will be generated by web server, communication happens through Rest/Soup UI Calls

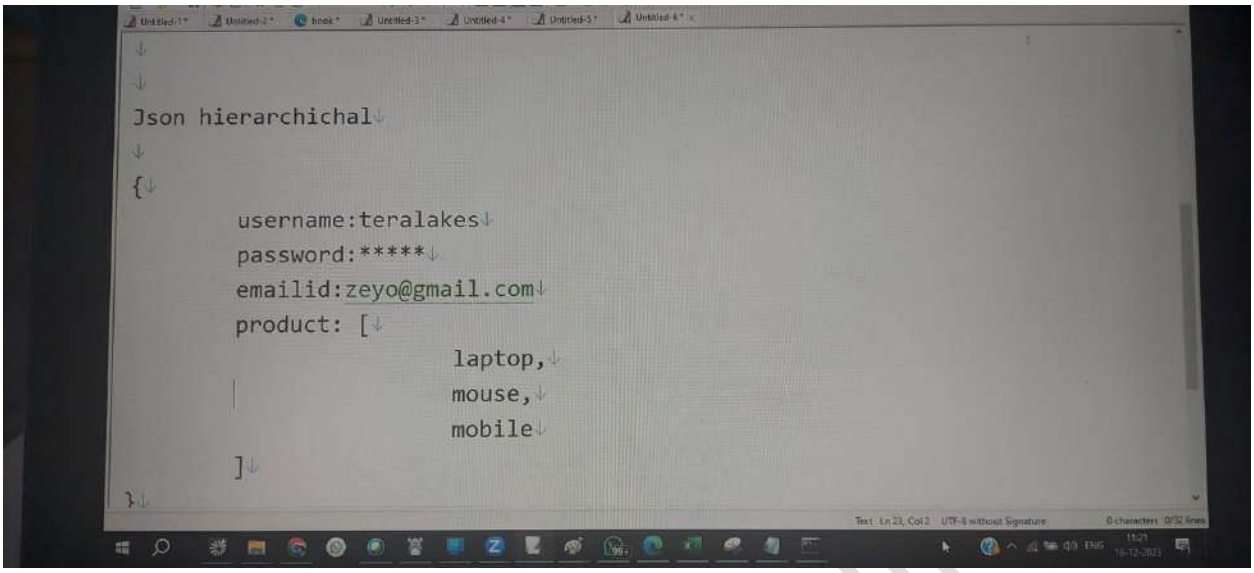




JSON, Parquet are popular in hierarchical data

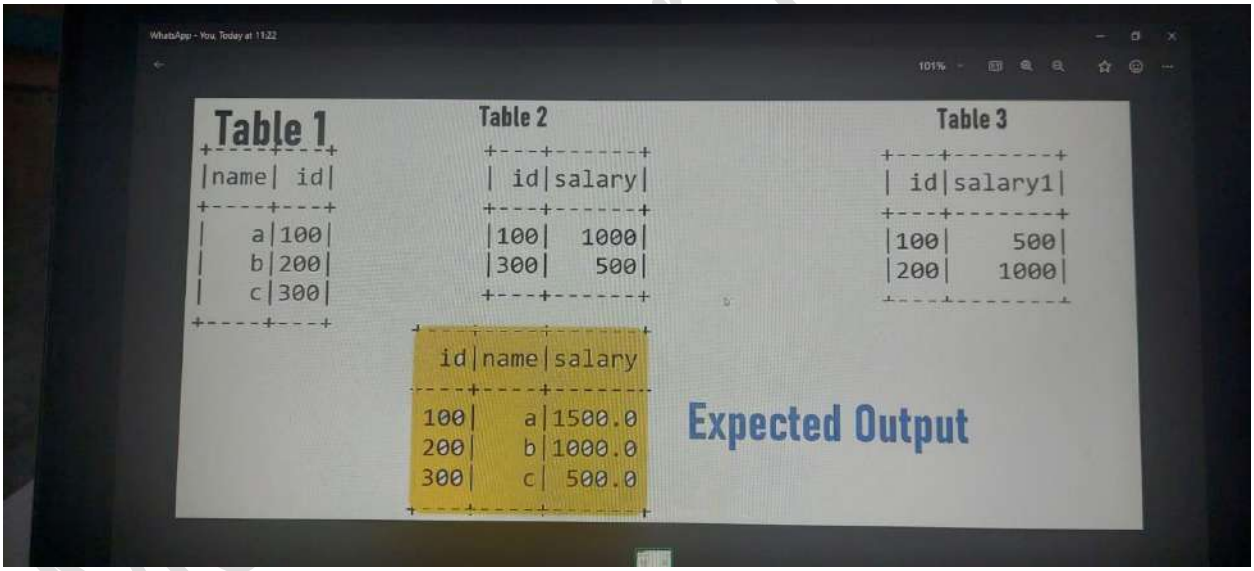
Hierarchical data is more suitable than flat data, because flat data has lot if duplicate data.





Task: Scenario

Data: d1.csv, d2.csv, d3.csv



Var joindf=Tab1.join(tab2, seq(“id”),”left”). join(tab3, seq(“id”),”left”)

//replace null with Zeros
var nonnull=joindf.withColumn(“salary”, expr(“coalesce(salary,0)”))
.withColumn(“salary1”, expr(“coalesce(salary1,0)”))

//Adding
Val sumdf= nonnull.withcolumn(“salary”, expr(“salary+salary1”))

Val finaldf= sumdf.selectExpt(“id”,”name”,”cast(salary as int) as salary”)
Finaldf.show()

Spark Complex Data Deep Dive Revision

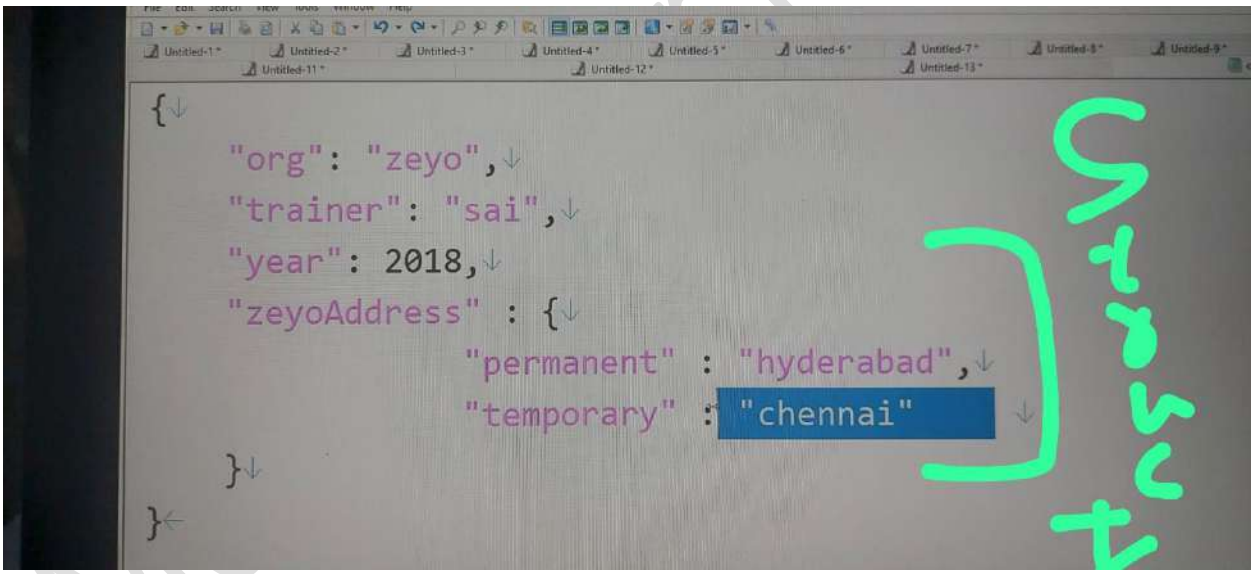
Agenda day 33

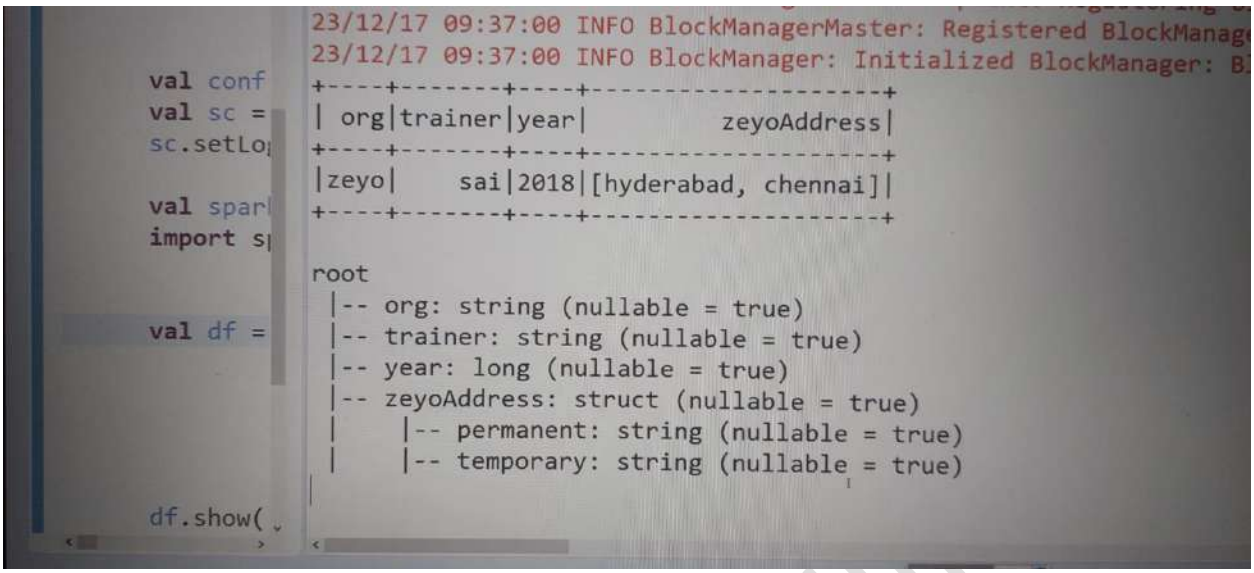
Complex Data deep dive

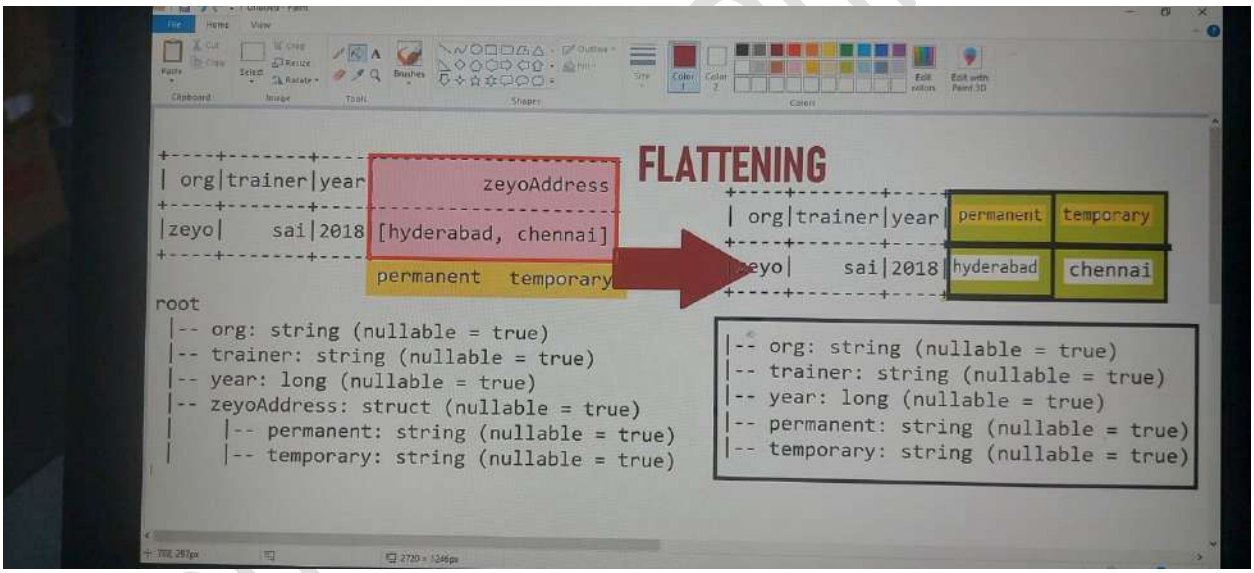
Complex data Type 1 and Type 2

When Json is in multiline, the spark couldn’t process that. For this, we need to set option(“multiline”, ”true”).

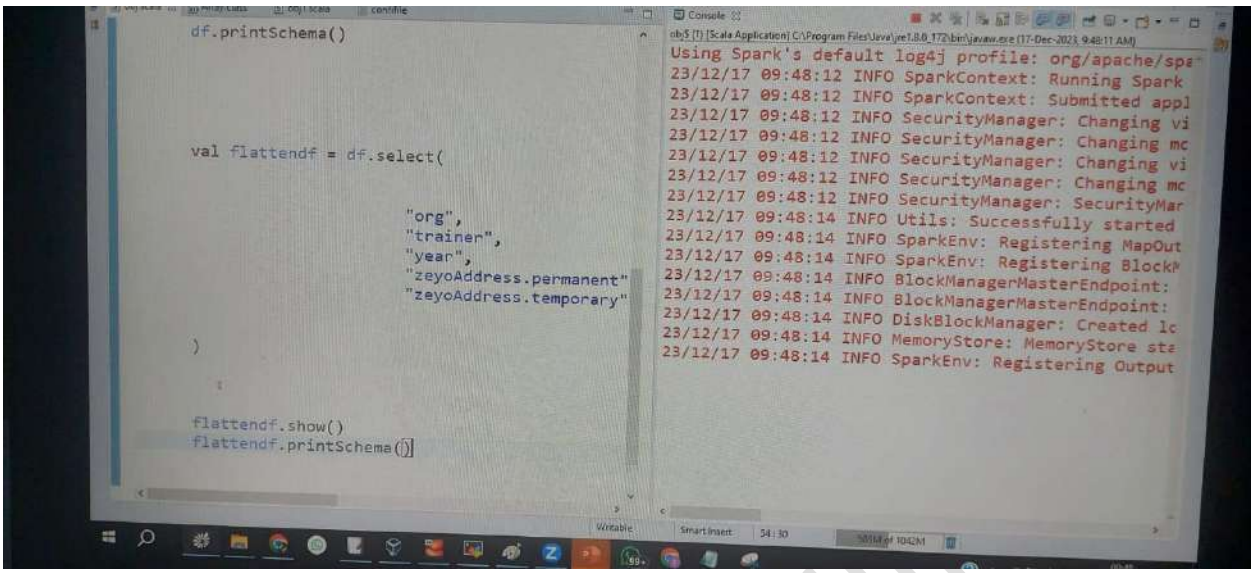
Simple JSON Validator: JSON Lint – JSON Validator website

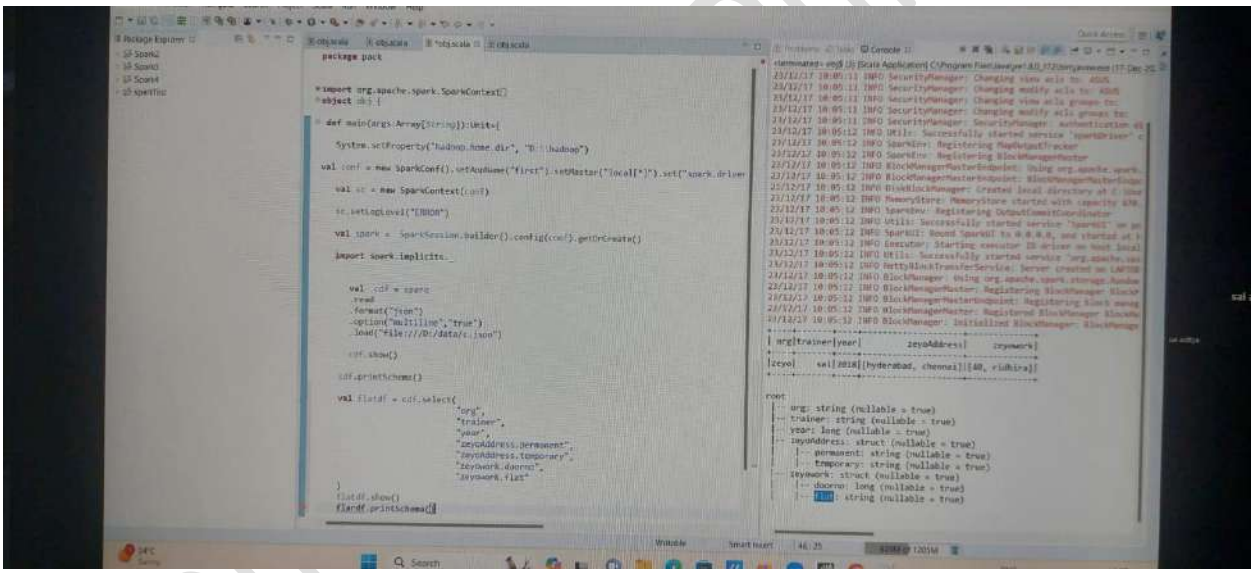






Select is used to convert and display hierarchical data to flat data, dot operator is used to flatten and read





Scala Tutorials:

https://www.youtube.com/watch?v=LQVDJtfpQU0&list=PLS1QulWo1RIagob5D6kMIAvu7DQC5VTh3

Tutorial 6 to 31

Revision

Collist = List[“Column1”,”column2”,”column3”]
val csvdf = spark.read.format(“csv”).load(<path>).select(collist.map(col):_*)

This is to convert the data to match collist when the data has different order of columns

val uniondf = schemadf.union(df1).union(df2)

➔ WithcolumnRenamed(oldcolumnname,newcolumnname) – to change the old column name to new column name.
df.withColumnRenamed(“txndata”,”year”)

Spark Complex data type 2 generation processing

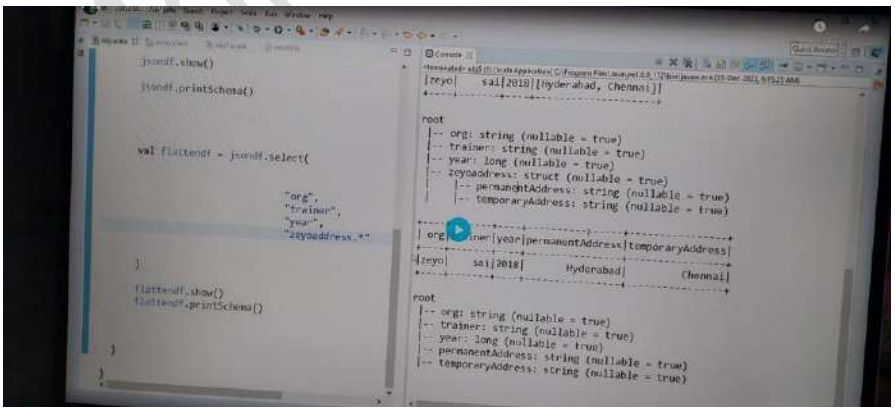
Agenda Day 34

Has to do:

Scala Tutorials for Beginners
Scenarios.
Interview Audios
Real Time Scenarios

Complex data Intro

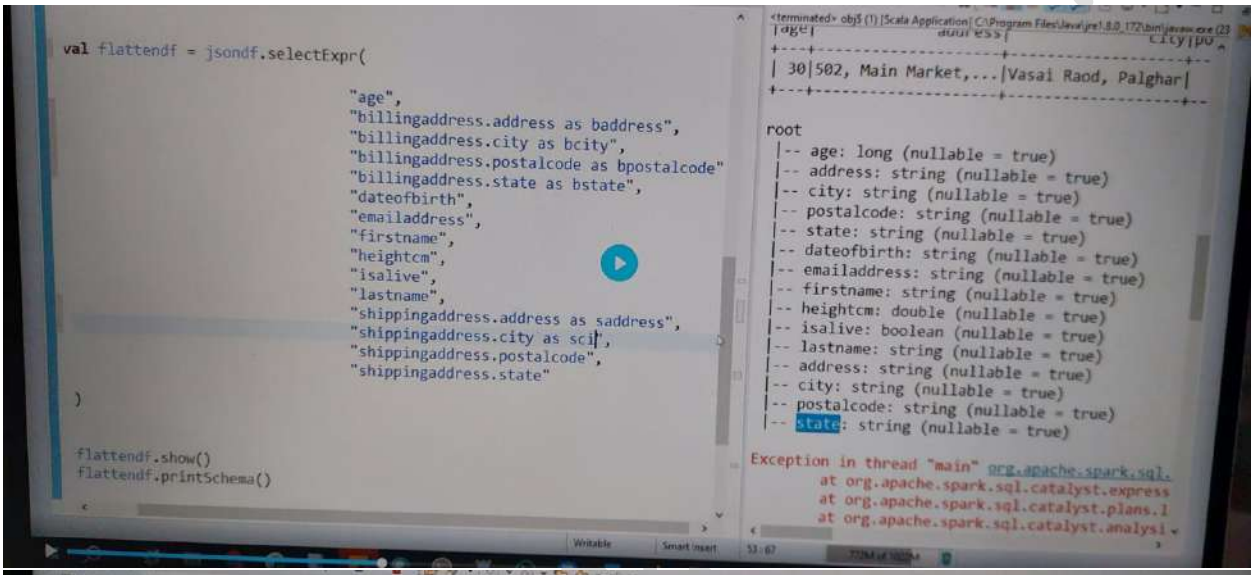
Complex data is hierarchical data.
Star can be used to print hierarchies on the same level.

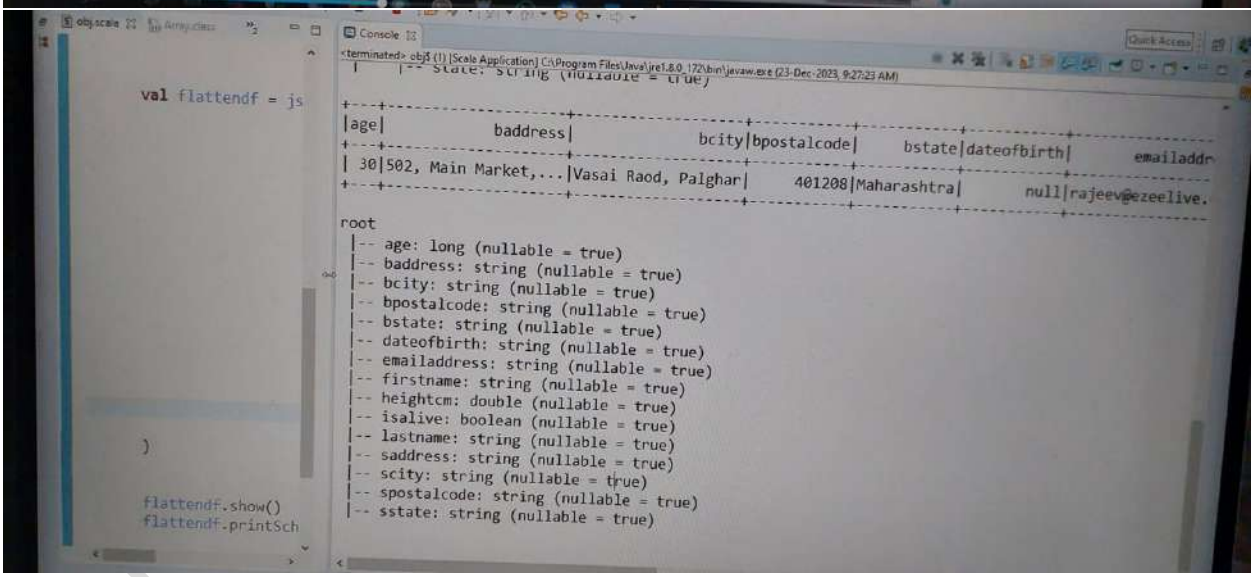


Complex data type2 handson

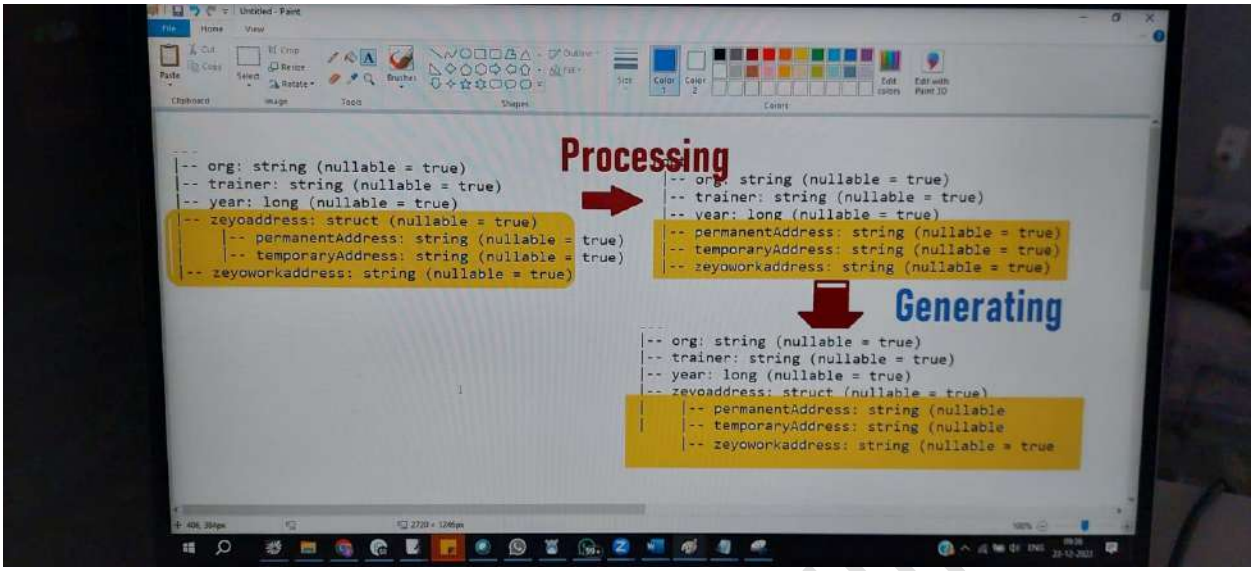
Ambiguity Exception:

on the Console, we see 2 addresses, one address is for shipping and the second address is for billing. So, when we flatten the data we need to rename them to a different name.

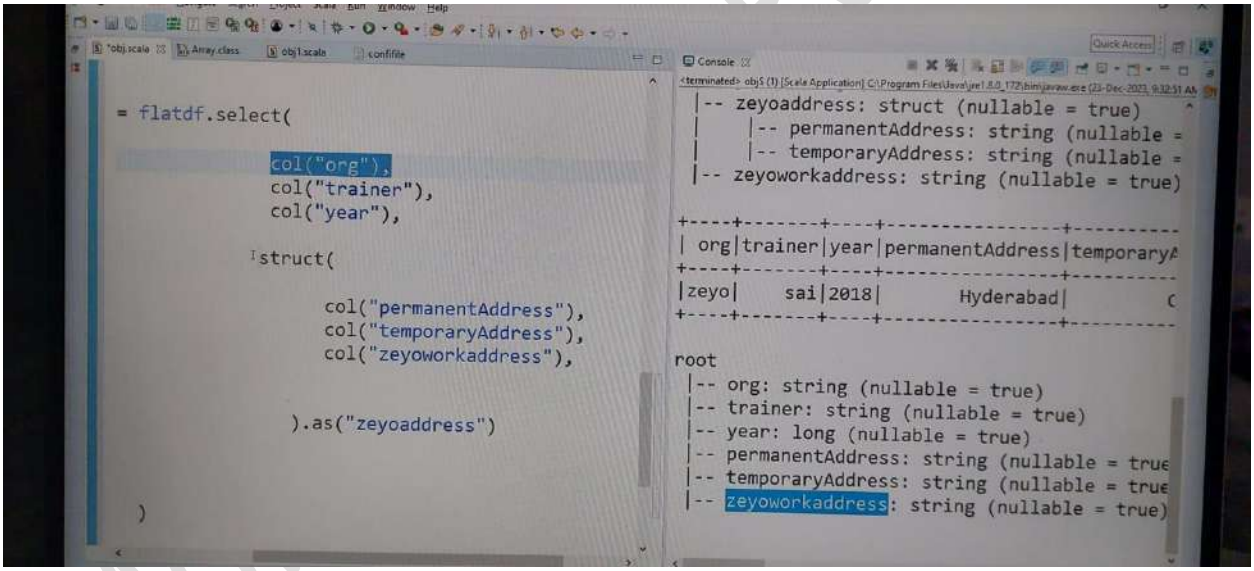


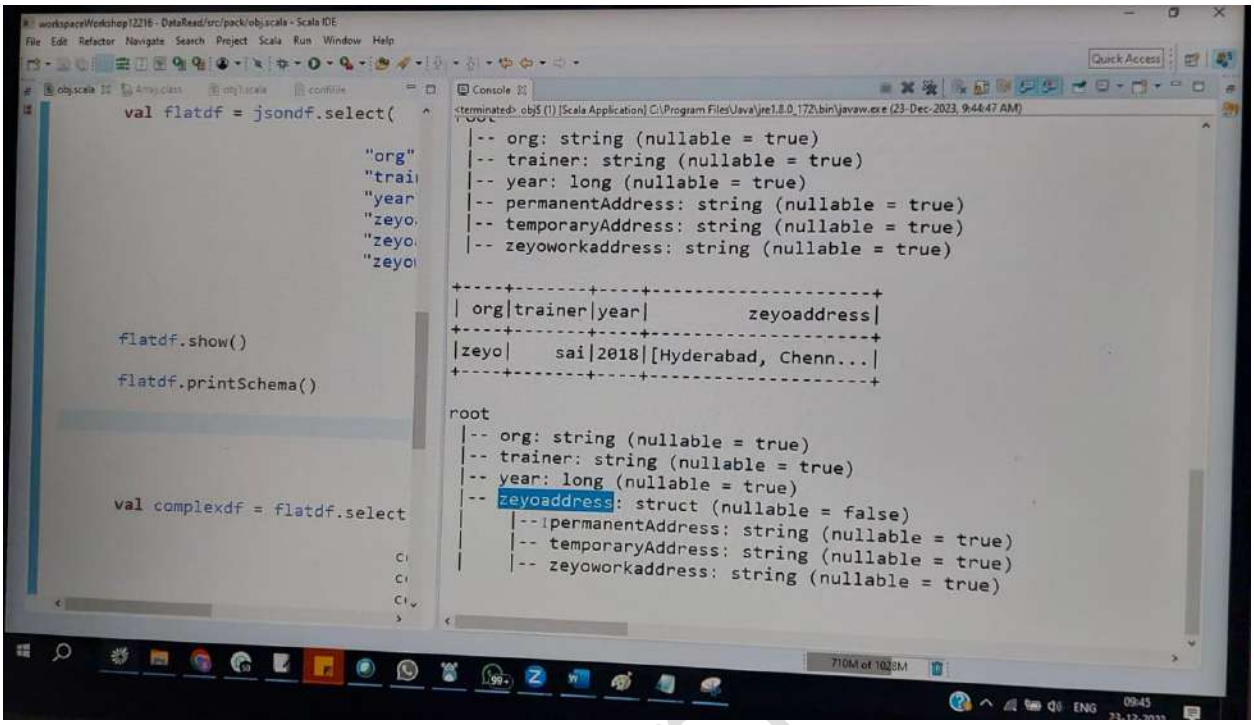


Complex data type 2 generation

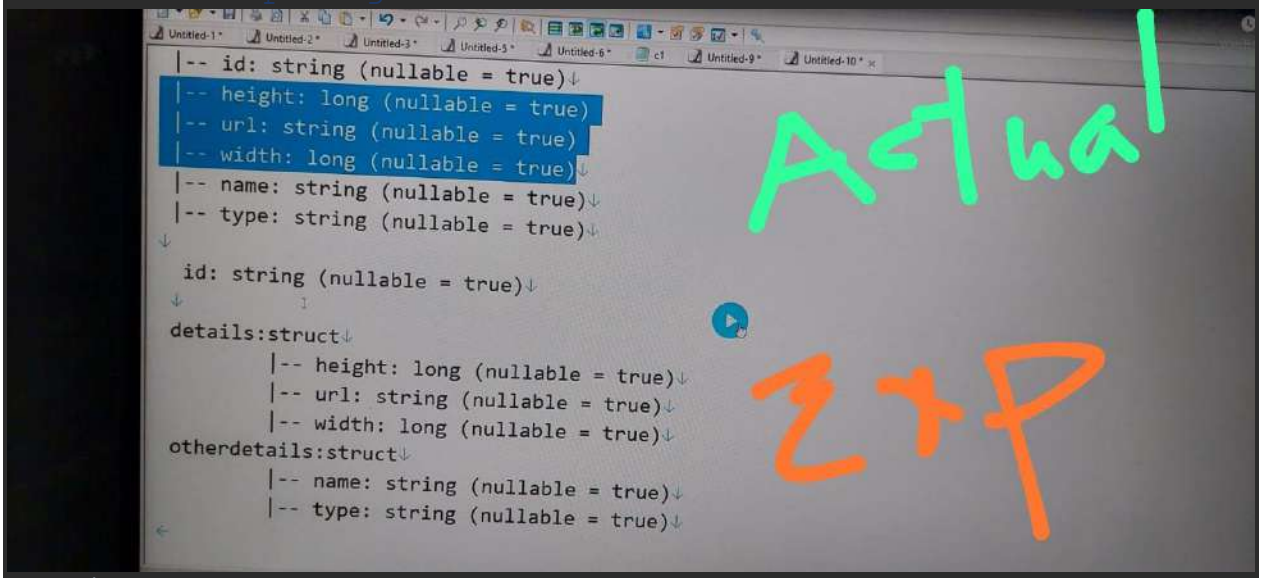


When we use function inside a side, we always have use col()





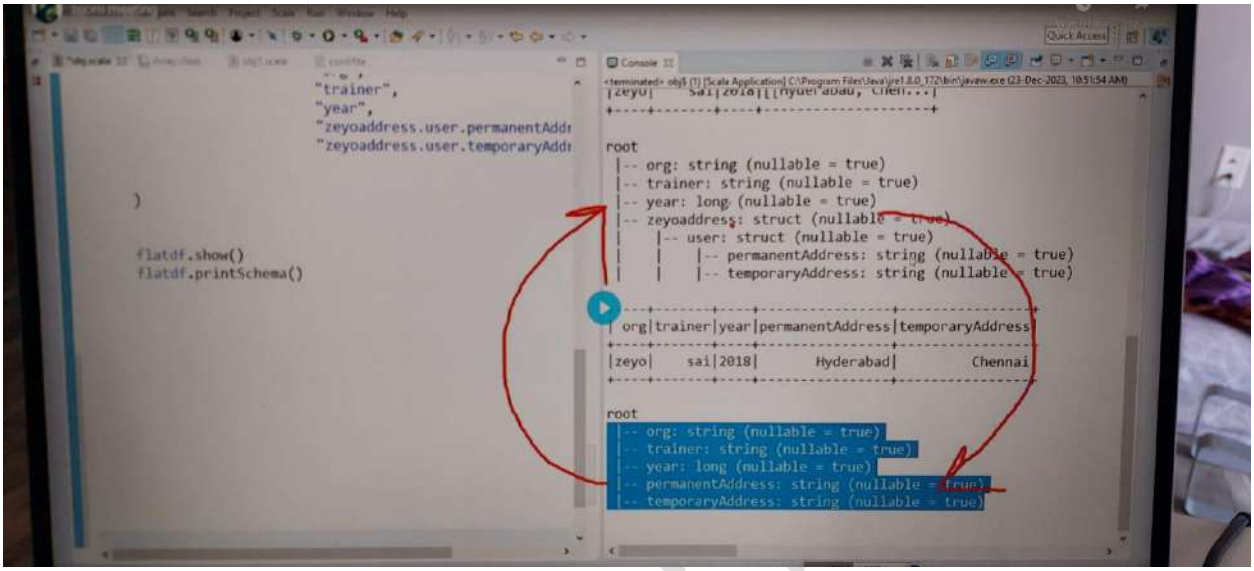
Practice: pic.json

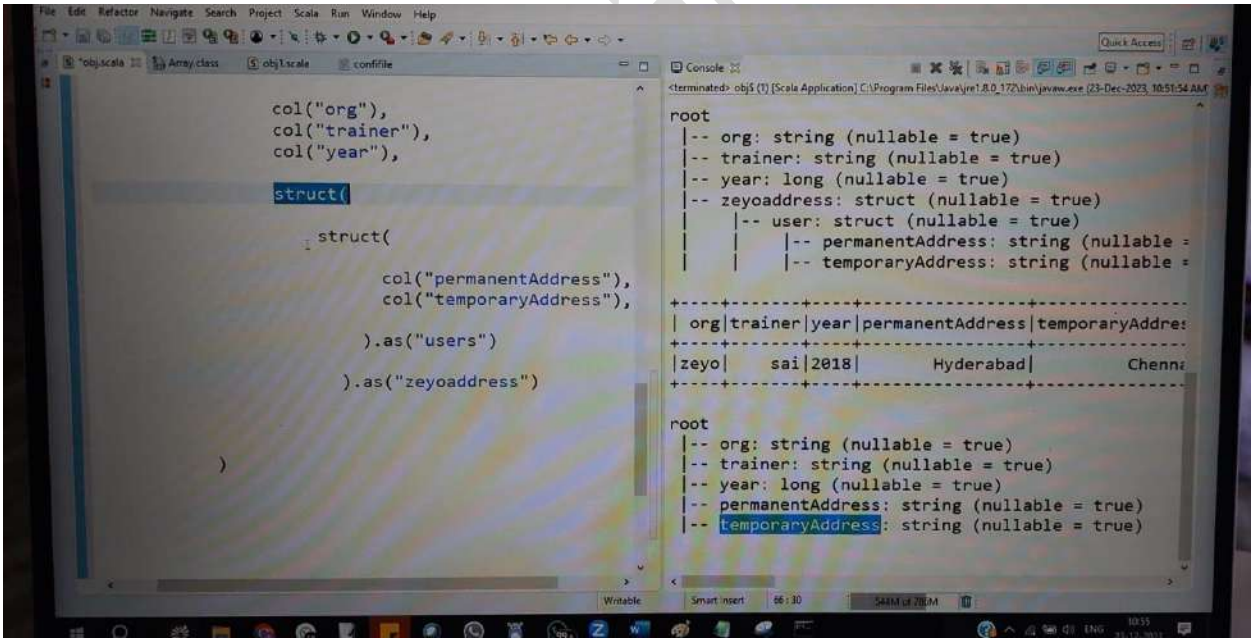



```
val jsondata =
spark.read.format("json").option("multiline","true").load("M:\\L
earnings\\Big_Data\\DataSets\\pic.json")
jsondata.show()
jsondata.printSchema()

val generateddf = jsondata.select(col("id"),
    struct(col("image.*")).as("details"),
```

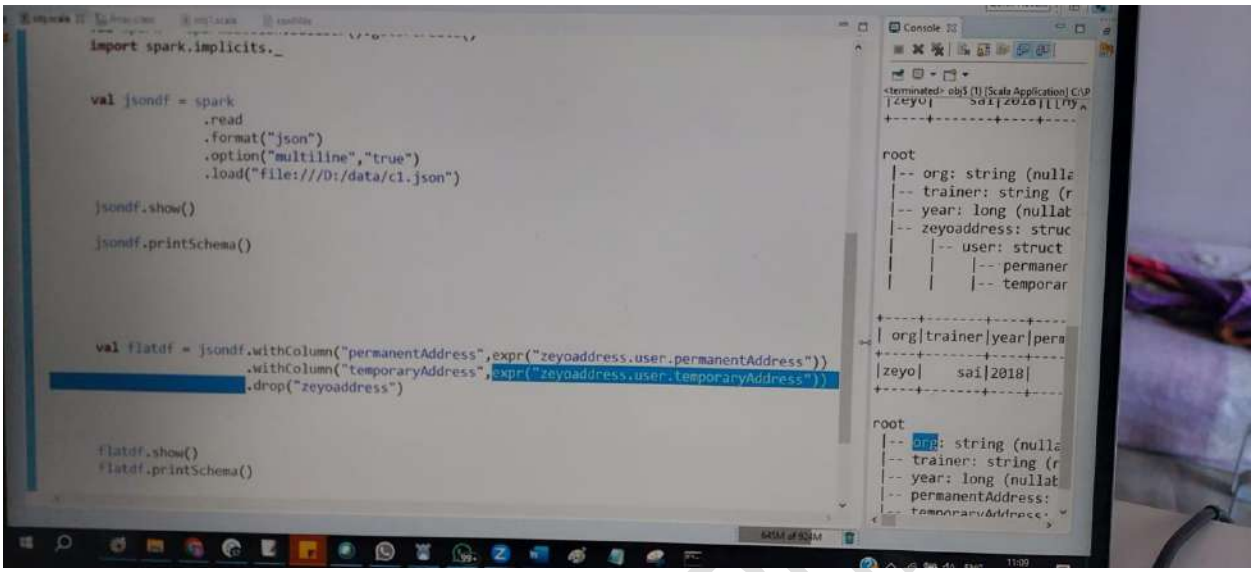
```
  struct(col("name"), col("type")).as("otherdetails") )

generateddf.printSchema()
```

Converting back to old schema

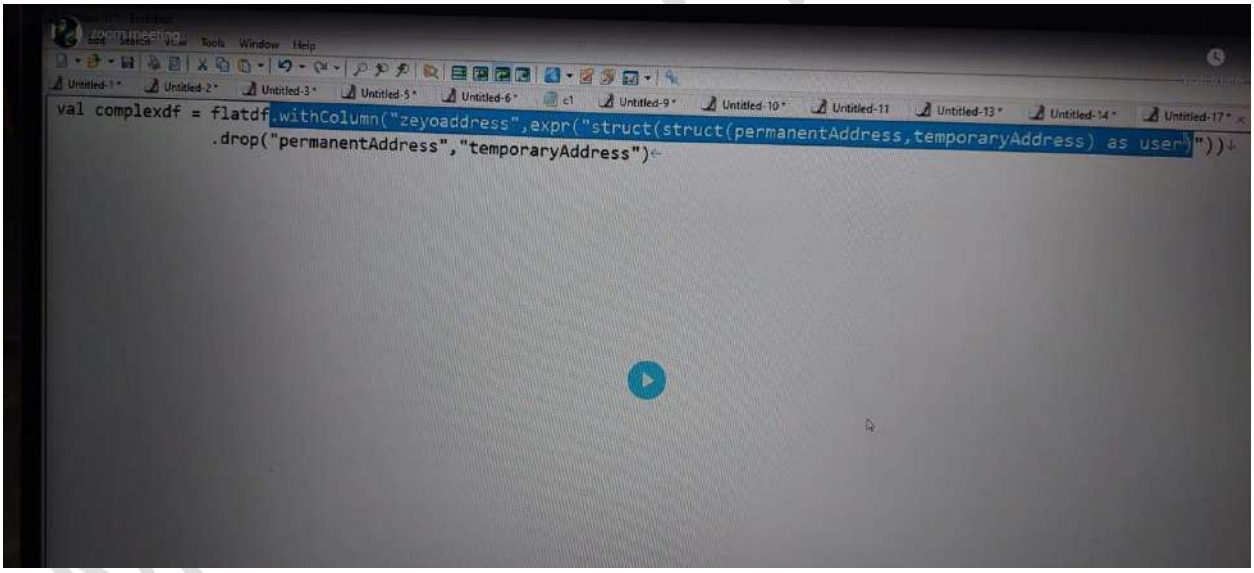




Flatten the struct using with column

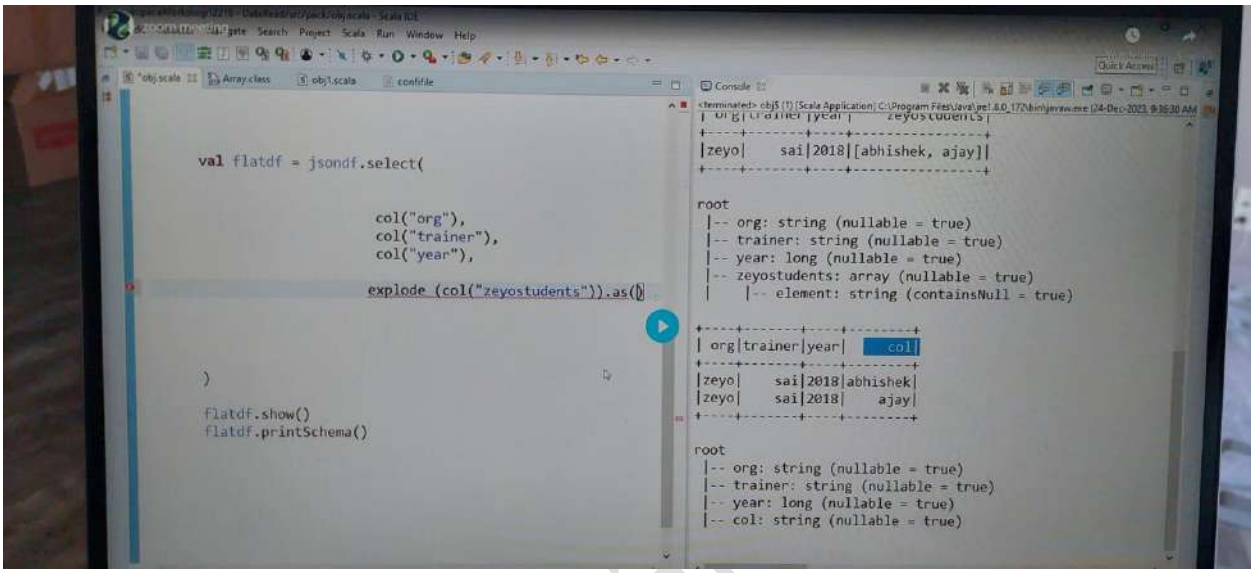


Converting to Hierarchy from flat data using with Column

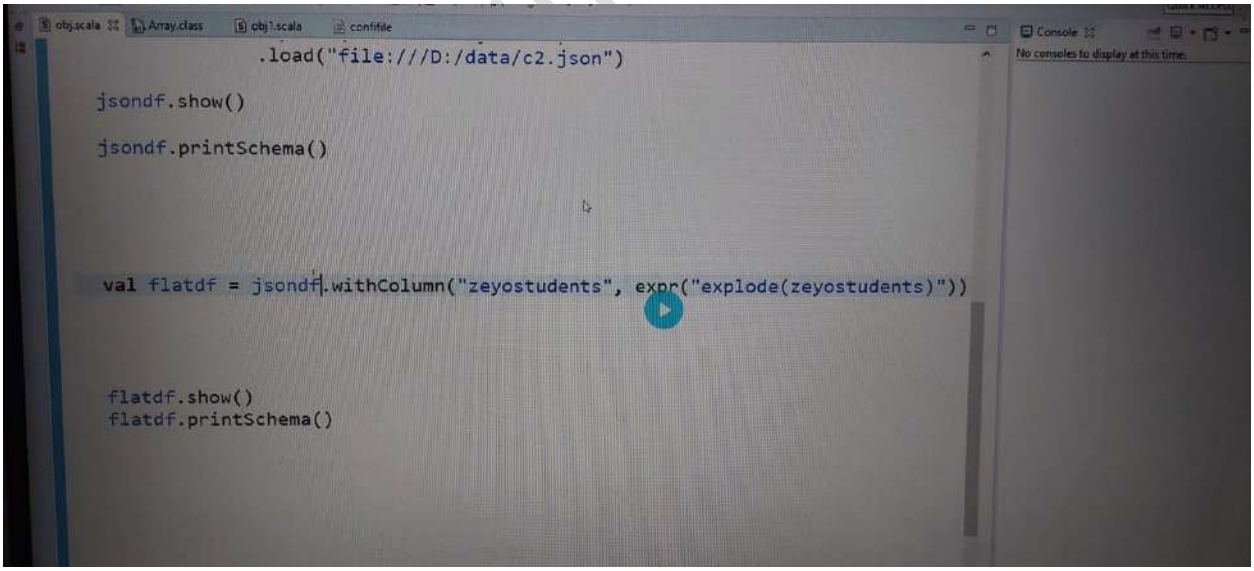


Spark Type 3 Array Type Resume Building
Agenda Day 35
Complex data type 3 processing Intro

Processing Arrays in JSON – using explode



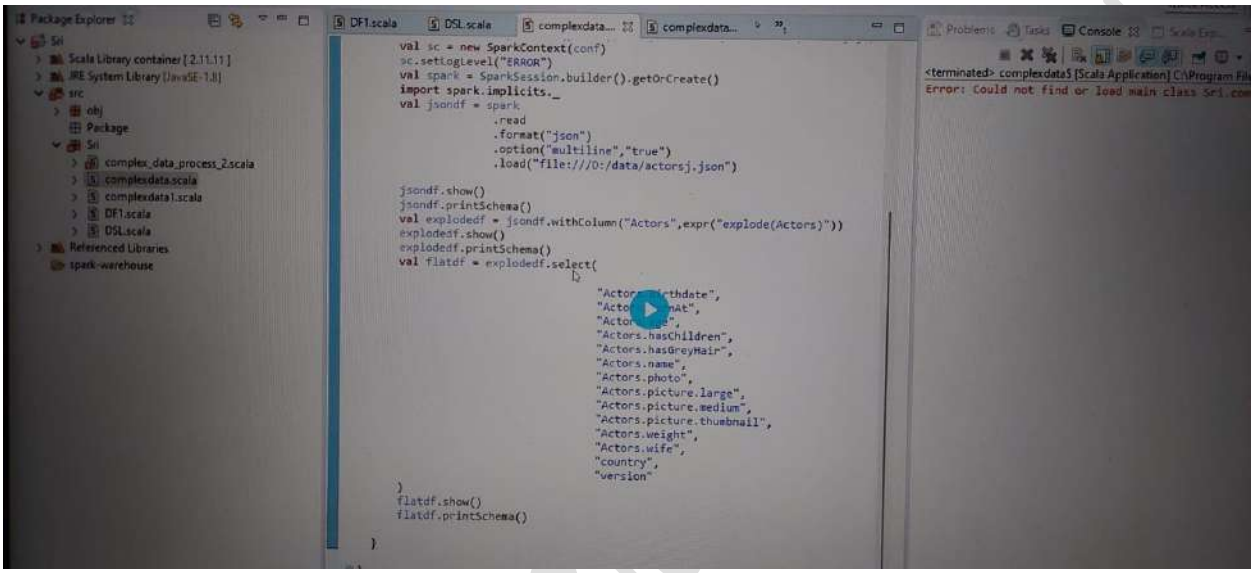
Processing Arrays in JSON – using withcolumn




```
val jsondata =
spark.read.format("json").option("multiline","true").load("M:\\L
earnings\\Big_Data\\DataSets\\pets1.json")
jsondata.printSchema()
```

```
jsondata.select(
  col("Boolean"),
      col("Mobile"),
  col("Name"),
  explode(col("Pets").as("Pets"))).show()
```

Another Scenario on explode: actorsj.json



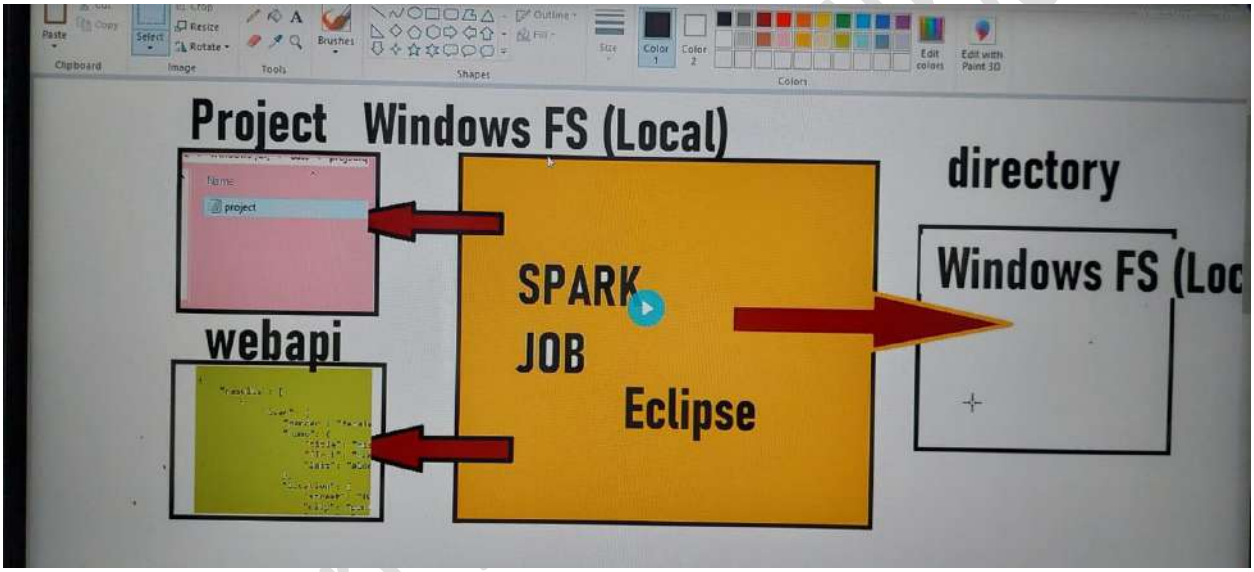
Getting the data from a URL:

```
val urldata =
Source.fromURL("https://randomuser.me/api/0.8/?results=
3").mkString

val df =
spark.read.json(sparContext.parallelize(List(urldata)))
df.show()
df.printSchema()
```

Complex data type3 Generation

Azure, GCP, Cloud Development Automation
Agenda day 36

Cloud Intro
Scenario intro
Scenario that we run in local and also in clouds.
Read the data from local file and webAPI, Flatten it, Remove numericals, Join, Write the data



Modify a column using regular expressions

```
df.withColumn("", regexp_replace(col(""),"[0-9]",""))
```

	aws	azure	gcp
Storage	S3	blob(storage account)	Cloud storage
Virtual Machine	EC2	VIRTUAL MACHINE	COMPUTE ENGINE
SQL	RDS	SQL DATABASES	CLOUD SQL
HADOOP	EMR	HDINSIGHT	DATAPROC
SERVERLESS	LAMBDA FUNCTION	AZURE FUNCTION	CLOUD FUNCTION
MQ	KINESIS	EVENT HUB	PUB/SUB

Cloud Development (GCP)

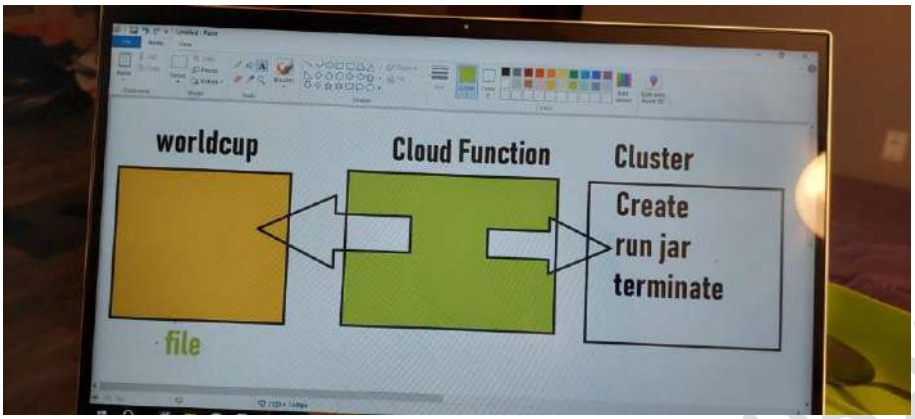
Steps:

1) Store the file in Google Cloud Storage
2) Create a Cluster in Dataproc
    3) Command to trigger in local command pmpt:
       gcloud compute ssh zeyo-cluster-m –zone asia-south2-a --
       After executing this command, Putty will automatically opens
    4) Spark-shell → this will open the spark shell
    5) Execute the same code that we have developed in our local system

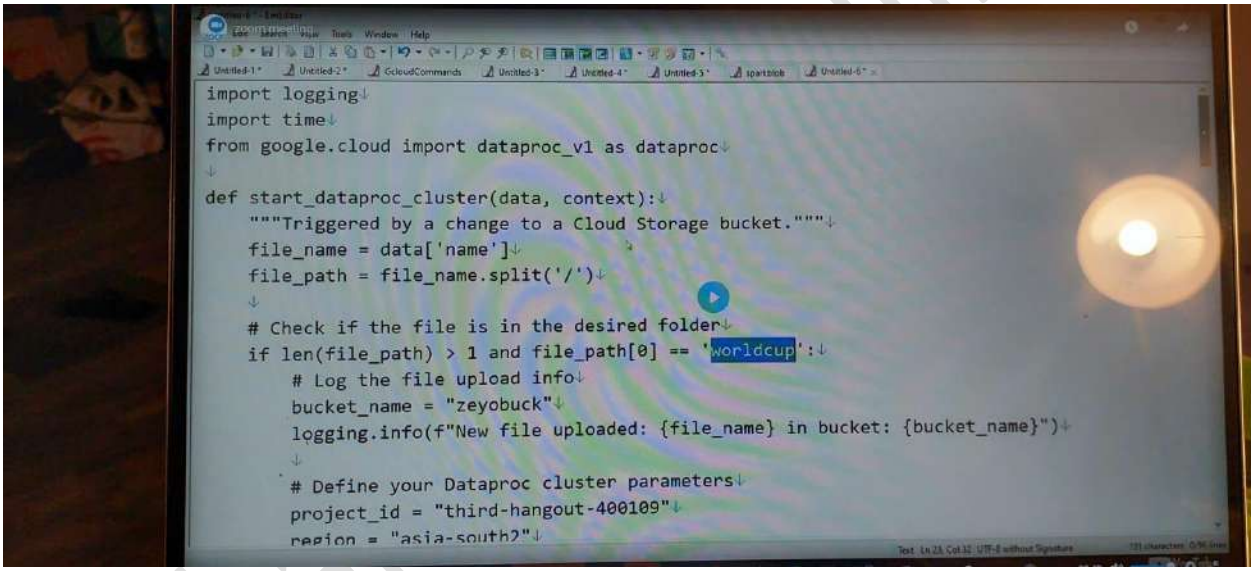
Now Executing this using Jar,

1) Complete development of the project in the Local and convert that to jar by executing maven build and package name as run parameter and get the jar file from target folder and upload that to GCP

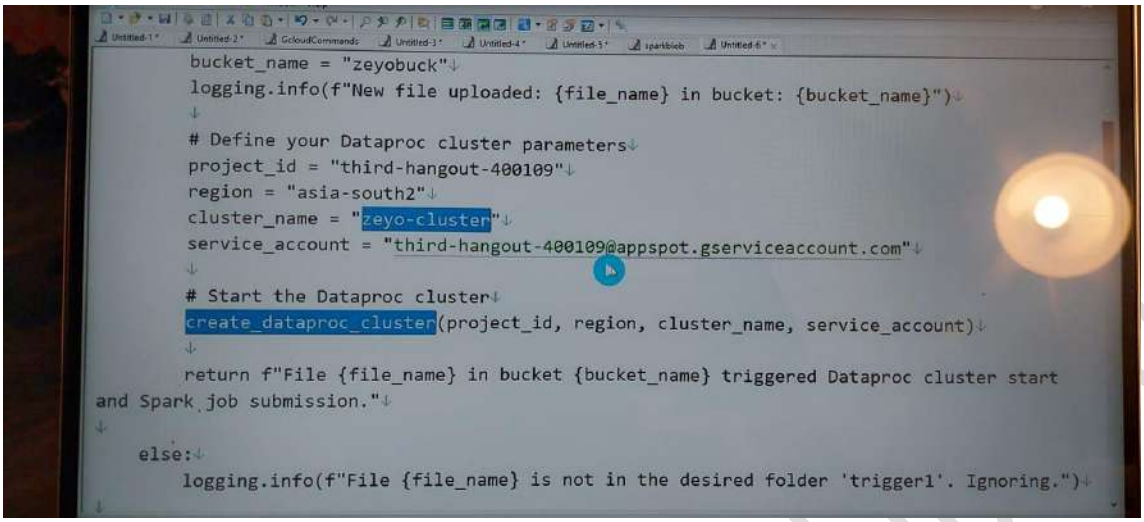
3) Open Cloud Function:



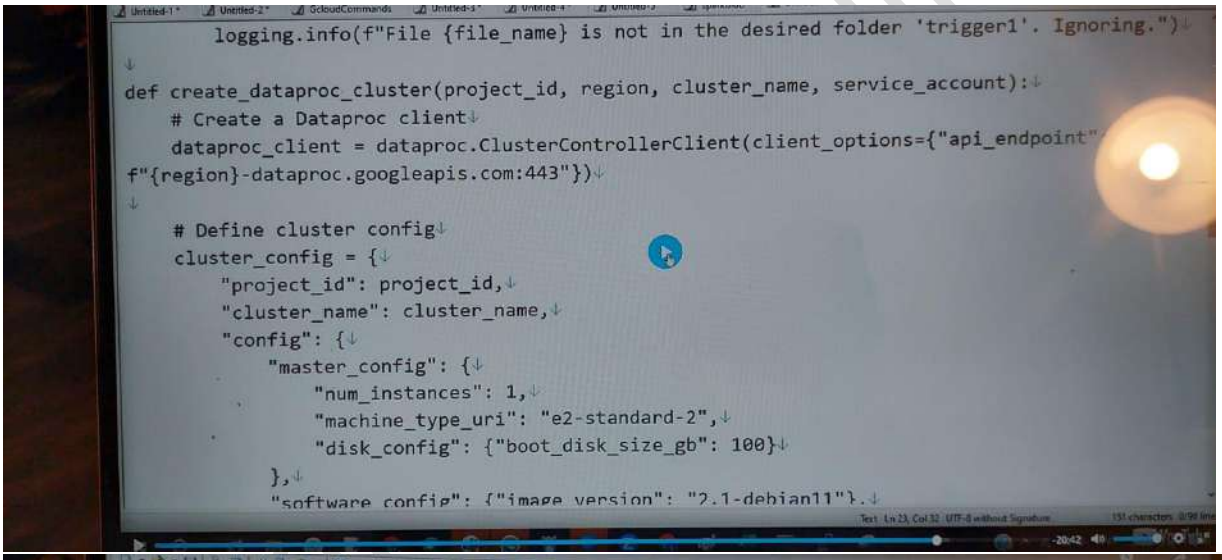
Cloud Function Code:



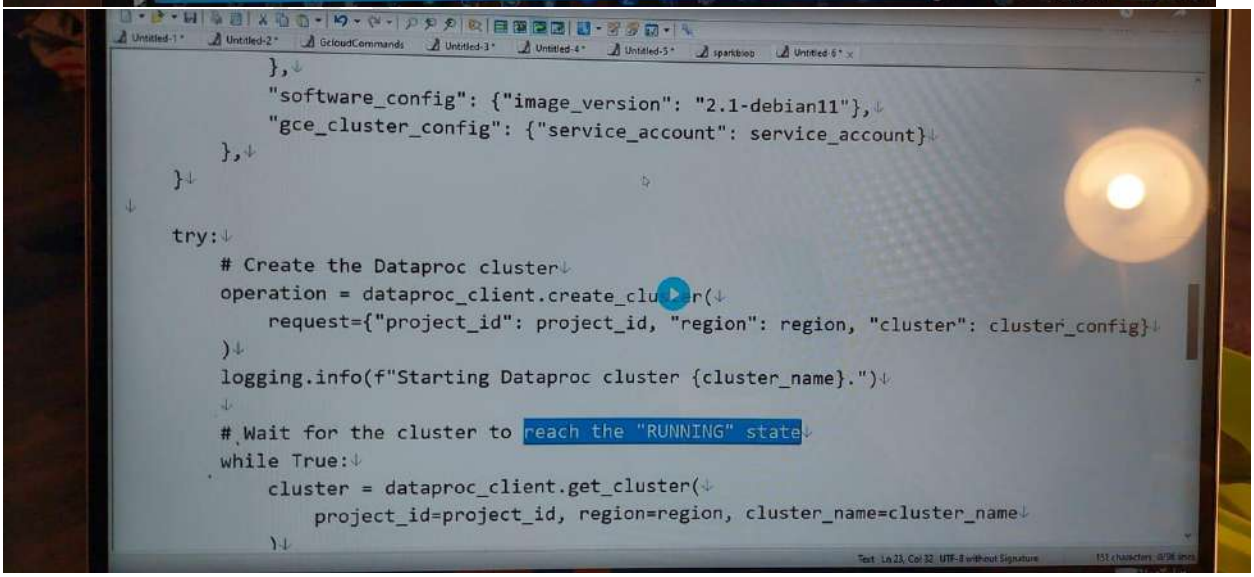
```
import logging
import time
from google.cloud import dataproc_v1 as dataproc

def start_dataproc_cluster(data, context):
    """Triggered by a change to a Cloud Storage bucket."""
    file_name = data['name']
    file_path = file_name.split('/')

    # Check if the file is in the desired folder
    if len(file_path) > 1 and file_path[0] == 'worldcup':
        # Log the file upload info
        bucket_name = "zeyobuck"
        logging.info(f"New file uploaded: {file_name} in bucket: {bucket_name}")

        # Define your Dataproc cluster parameters
        project_id = "third-hangout-400109"
        region = "asia-south2"
```



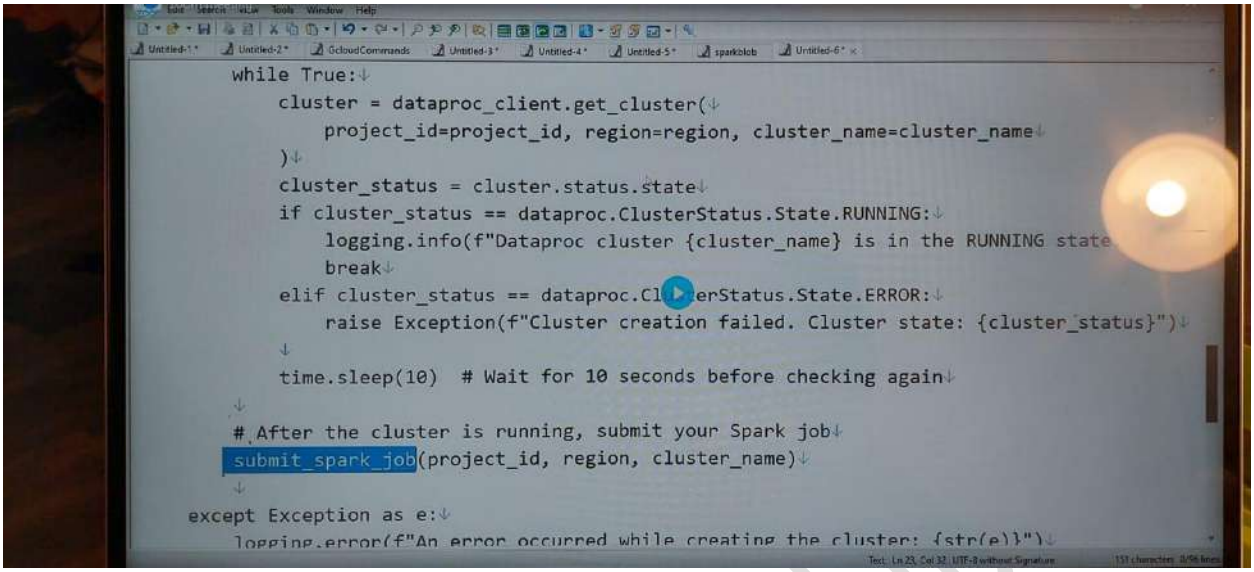
```
        bucket_name = "zeyobuck"
        logging.info(f"New file uploaded: {file_name} in bucket: {bucket_name}")

        # Define your Dataproc cluster parameters
        project_id = "third-hangout-400109"
        region = "asia-south2"
        cluster_name = "zeyo-cluster"
        service_account = "third-hangout-400109@appspot.gserviceaccount.com"

        # Start the Dataproc cluster
        create_dataproc_cluster(project_id, region, cluster_name, service_account)

        return f"File {file_name} in bucket {bucket_name} triggered Dataproc cluster start
and Spark job submission."

    else:
        logging.info(f"File {file_name} is not in the desired folder 'trigger1'. Ignoring.")
```



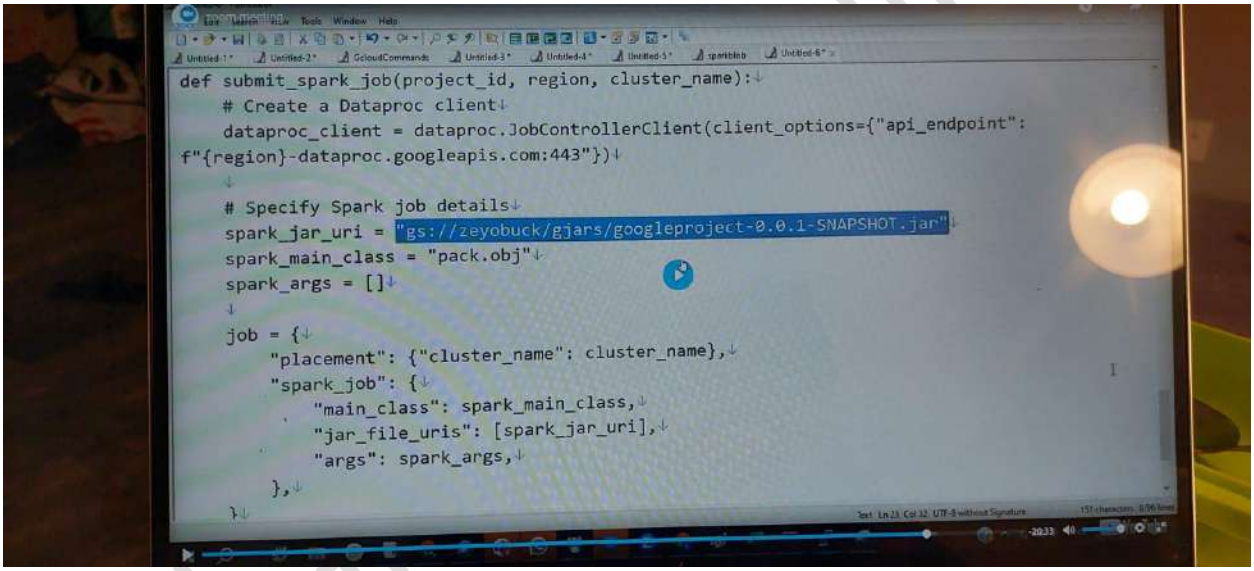
```
        logging.info(f"File {file_name} is not in the desired folder 'trigger1'. Ignoring.")

def create_dataproc_cluster(project_id, region, cluster_name, service_account):
    # Create a Dataproc client
    dataproc_client = dataproc.ClusterControllerClient(client_options={"api_endpoint"
f"{region}-dataproc.googleapis.com:443"})

    # Define cluster config
    cluster_config = {
        "project_id": project_id,
        "cluster_name": cluster_name,
        "config": {
            "master_config": {
                "num_instances": 1,
                "machine_type_uri": "e2-standard-2",
                "disk_config": {"boot_disk_size_gb": 100}
            },
            "software_config": {"image_version": "2.1-debian11"},
```



```
            },
            "software_config": {"image_version": "2.1-debian11"},
            "gce_cluster_config": {"service_account": service_account}
        },
    }

    try:
        # Create the Dataproc cluster
        operation = dataproc_client.create_cluster(
            request={"project_id": project_id, "region": region, "cluster": cluster_config}
        )
        logging.info(f"Starting Dataproc cluster {cluster_name}.")

        # Wait for the cluster to reach the "RUNNING" state
        while True:
            cluster = dataproc_client.get_cluster(
                project_id=project_id, region=region, cluster_name=cluster_name
            )
```



```
        while True:
            cluster = dataproc_client.get_cluster(
                project_id=project_id, region=region, cluster_name=cluster_name
            )
            cluster_status = cluster.status.state
            if cluster_status == dataproc.ClusterStatus.State.RUNNING:
                logging.info(f"Dataproc cluster {cluster_name} is in the RUNNING state
                break
            elif cluster_status == dataproc.Cl erStatus.State.ERROR:
                raise Exception(f"Cluster creation failed. Cluster state: {cluster_status}")

            time.sleep(10)  # Wait for 10 seconds before checking again

        # After the cluster is running, submit your Spark job
        submit_spark_job(project_id, region, cluster_name)

    except Exception as e:
        logging.error(f"An error occurred while creating the cluster: {str(e)}")
```



```
def submit_spark_job(project_id, region, cluster_name):
    # Create a Dataproc client
    dataproc_client = dataproc.JobControllerClient(client_options={"api_endpoint":
f"{region}-dataproc.googleapis.com:443"})

    # Specify Spark job details
    spark_jar_uri = "gs://zeyobuck/gjars/googleproject-0.0.1-SNAPSHOT.jar"
    spark_main_class = "pack.obj"
    spark_args = []

    job = {
        "placement": {"cluster_name": cluster_name},
        "spark_job": {
            "main_class": spark_main_class,
            "jar_file_uris": [spark_jar_uri],
            "args": spark_args,
        },
    }
```

The same can be achieved using data factory by creating a pipeline in Azure

Cloud Development (Azure)

Steps:

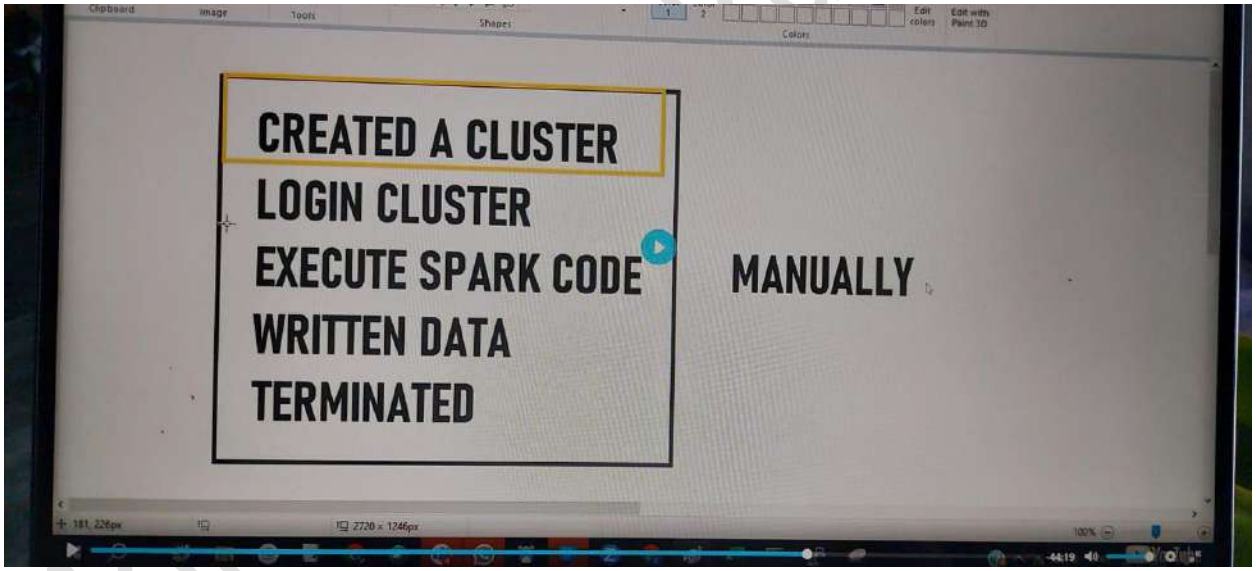
1) Store the file in Blob Storage
2) Create a Cluster in HDInsights
    4) Open the Cluster in HDinsights, go to SSH+Cluster login and get the username and password and connect to the cluster using putty

5) Designing the read path of the file on Azure Blob
for example path of the file is ZeyoClusterhdiStorage → hdiplaceholder →project.parquest
wasbs://hdiplaceholder@ZeyoClusterhdiStorage.blob.core.windows.net/project.parquest

wasbs → windows Azure storage blobs

→Save path looks as below:
wasbs://
hdiplaceholder@ZeyoClusterhdiStorage.blob.core.windows.net/azuredest

In the above execution, we have done everything manually, but in the real world we won't do everything manually

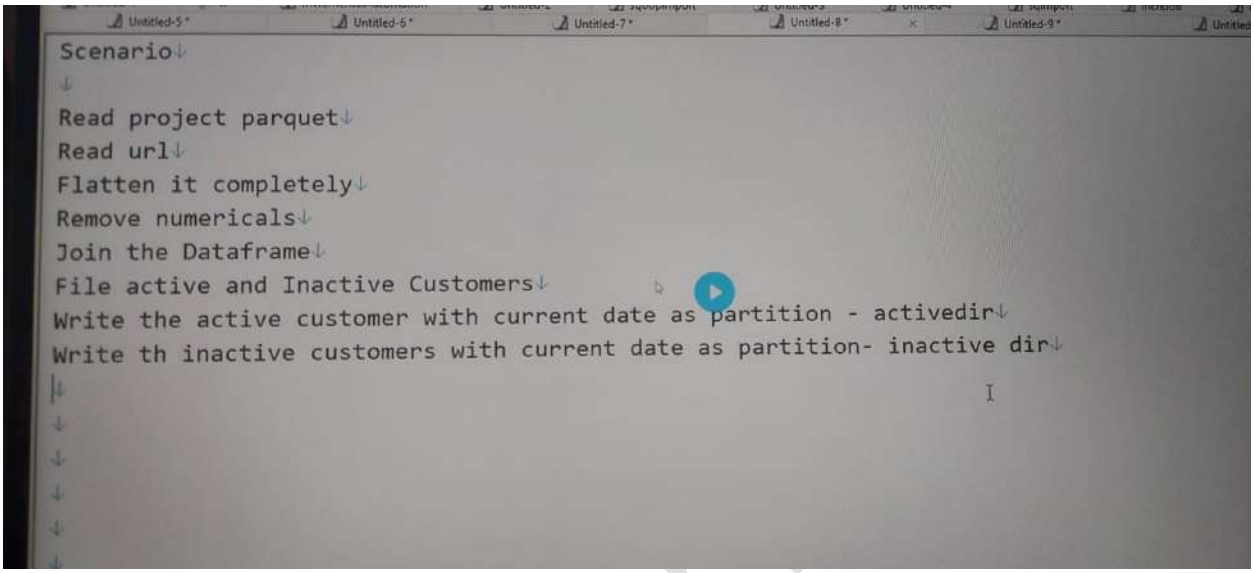


To Automate this, Cloud Function in GCP and Data factory in azure, comes into picture.

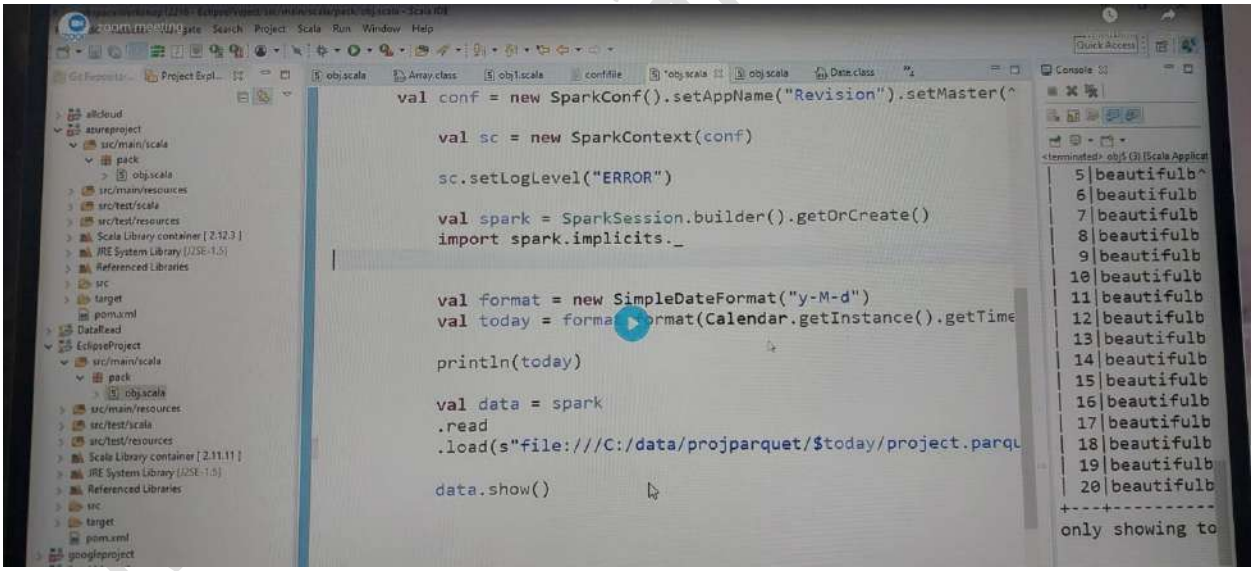
Cloud Deployment (GCP)
Cloud Deployment (Azure)

Spark GCP Deep Dive Cloud Function
Agenda Day 37

Scenario – Project Info



String Interpolation in Scala(Read Project):

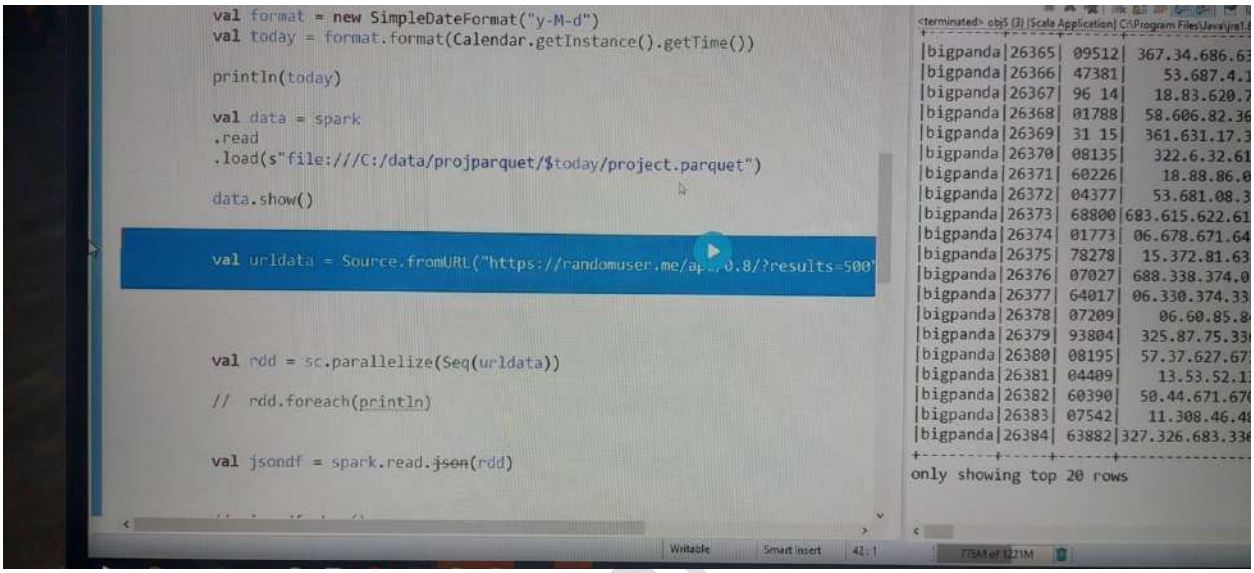


Reading Data from URL:
val urldata= Source.fromURL("<URL>")

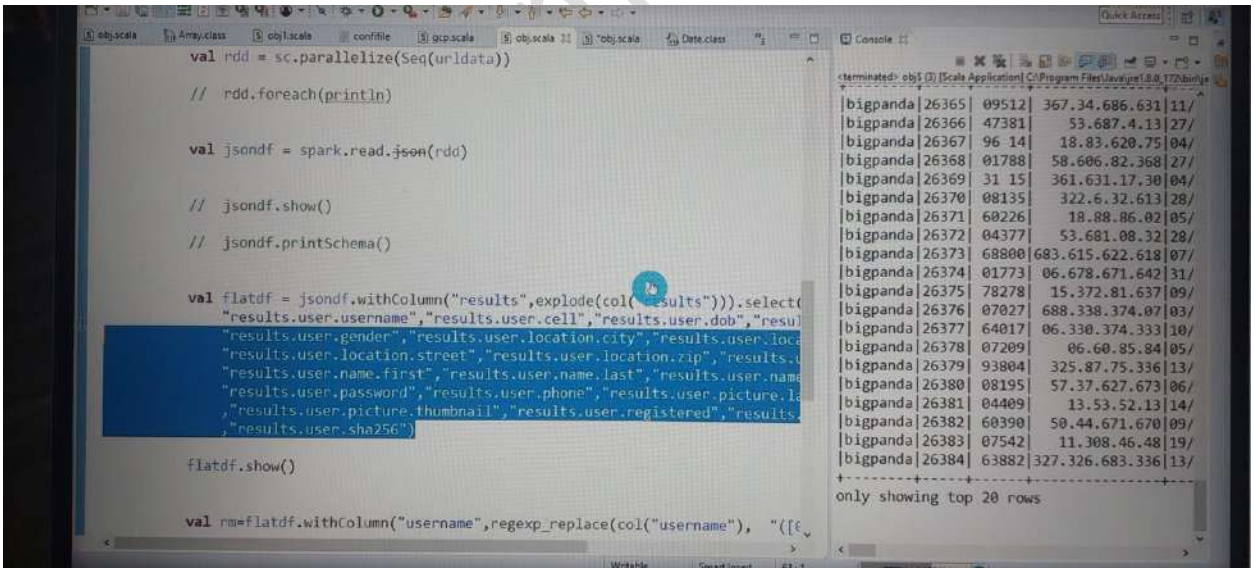
Converting String to RDD:
val rdd = sc.parallelize(Seq(urldata)) → sc here springcontext

Converting rdd json to DF
val jsondf = spark.read.json(rdd)
Jsondf.show()

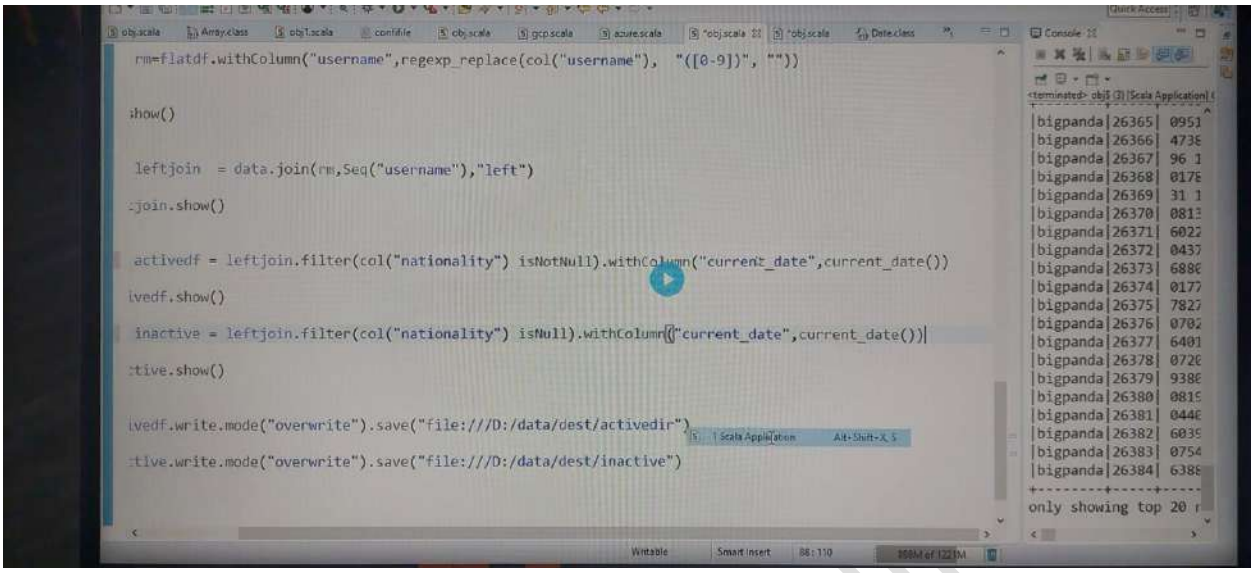
Read URL:



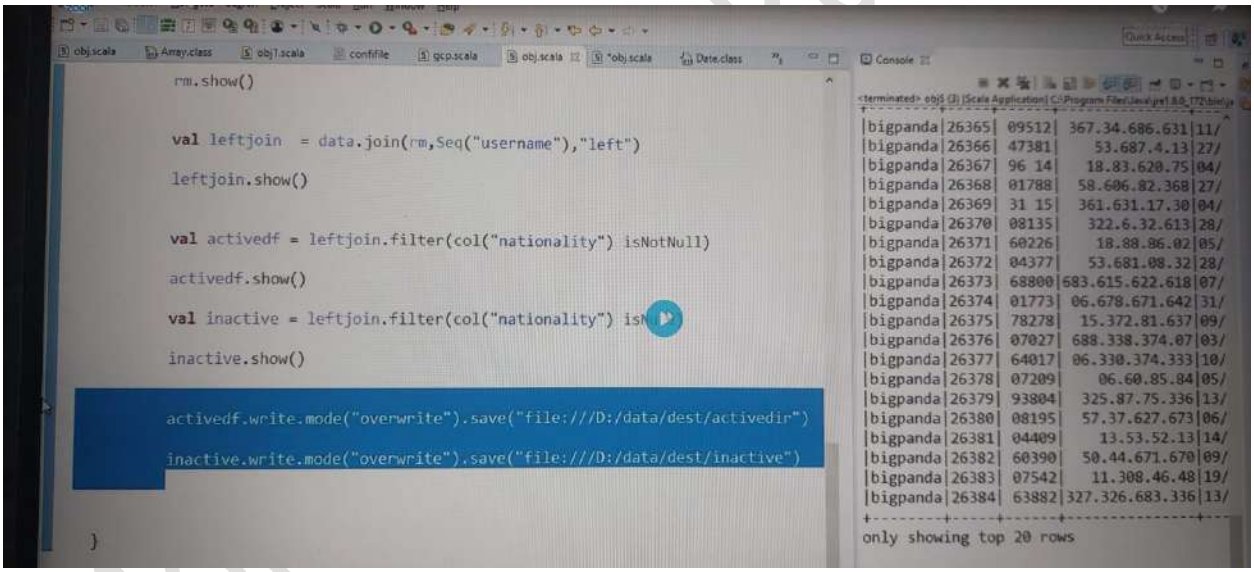
Flatten the data:



Remove Numerical in the column username:



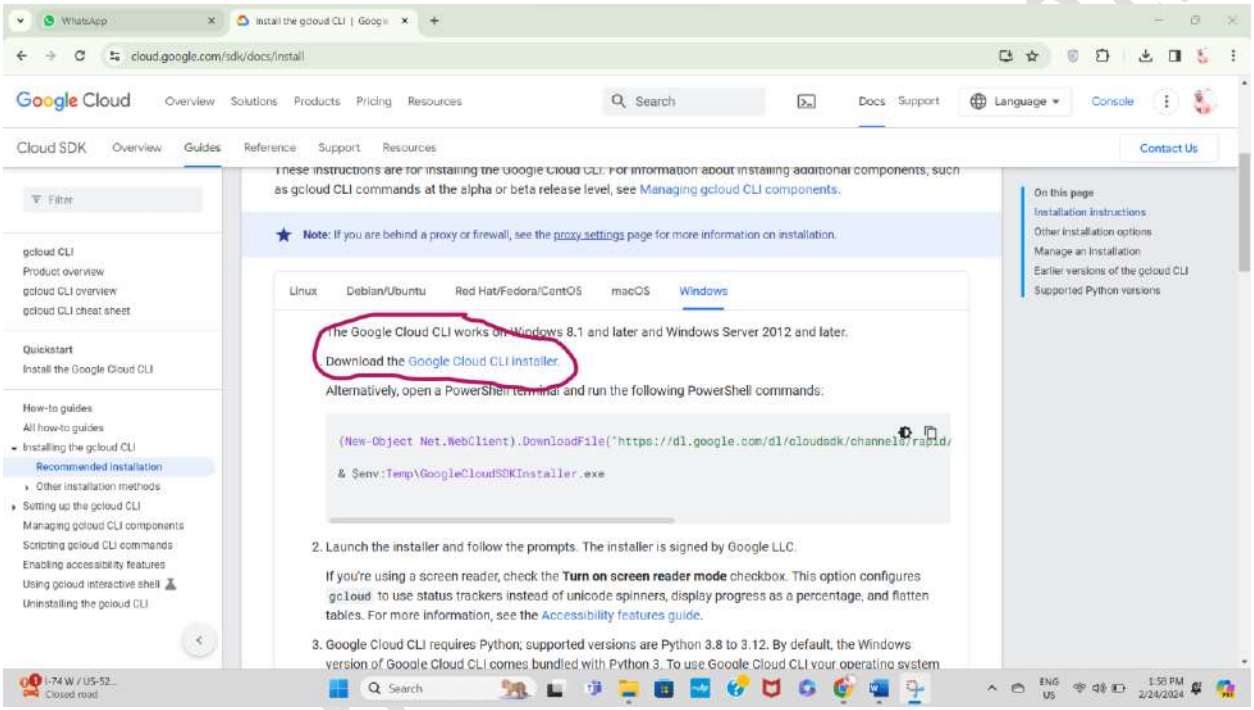
Write as parquet and save them to a directory:



Deep into Google Pipeline

Create a dataproc cluster, and a bucket in google storage.

To Login to the dataproc cluster:
1) Install gcloud SDK
https://cloud.google.com/sdk/docs/install



gcloud version in command prompt is to check the version of gcloud installed

2) Get authenticated

In command Prompt:
gcloud auth login
choose the account and allow

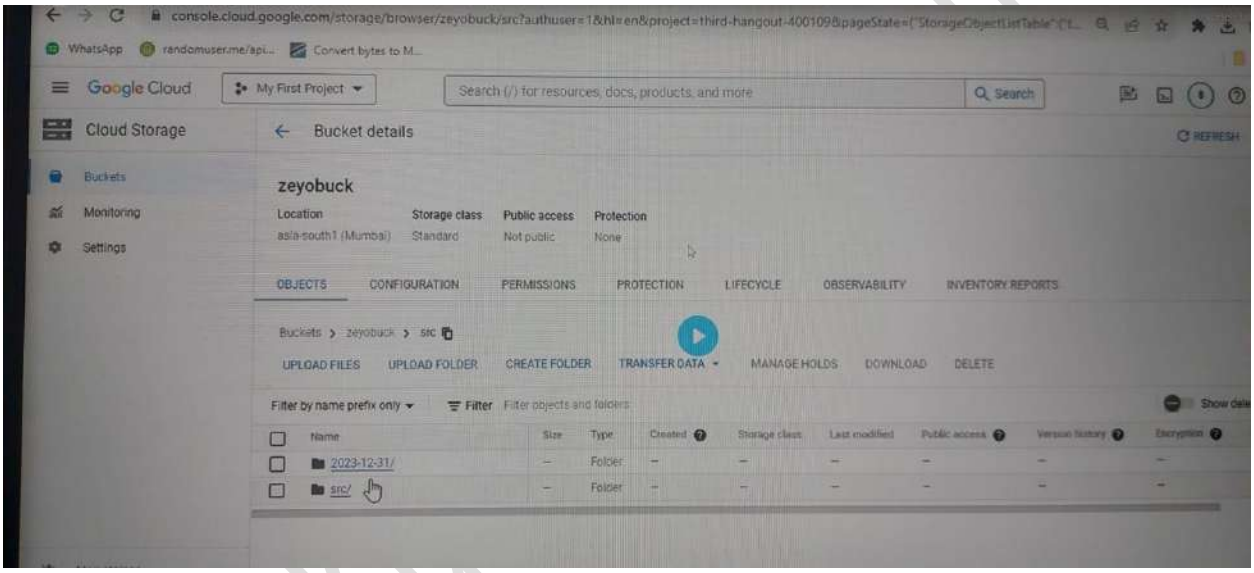
3) Login to cluster using a command

gcloud compute ssh zeyo-cluster-m --zone asia-south2-a --

gcloud compute ssh <Cluster_Name>-m --zone asia-south2-a -- (m stands for master node)

this takes us to the putty.

Create a bucket in GCP and use that:



Open Spark-shell in the putty opened in Step-3 and execute the spark code.
use the above spark-code developed by just changing the paths.

Automating the above process:

we can do the above process using Google CLI, Cloud Functions, Kubernetes Automation.

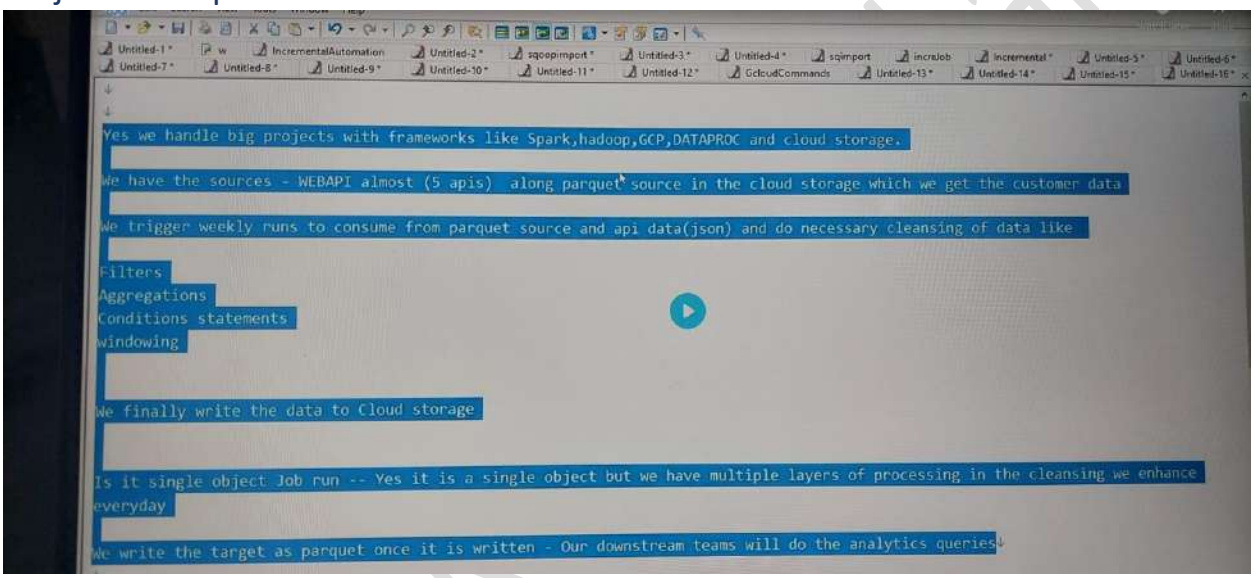
AWS → Lambda Function
Google → Cloud Function
Azure → Azure Functions

All these services are for event based triggers(Eg: Upload a file to a storage is an

event trigger).
Cloud Functions keep listening to these Events and performs the actions.

Creating Cloud Function:
1) Choose Environment, Give Function name, Select region, Choose type of trigger, Choose bucket if it's a storage trigger.
2) Choose the runtime (Java, Node etc)
Choose the main function in your code and paste the code.

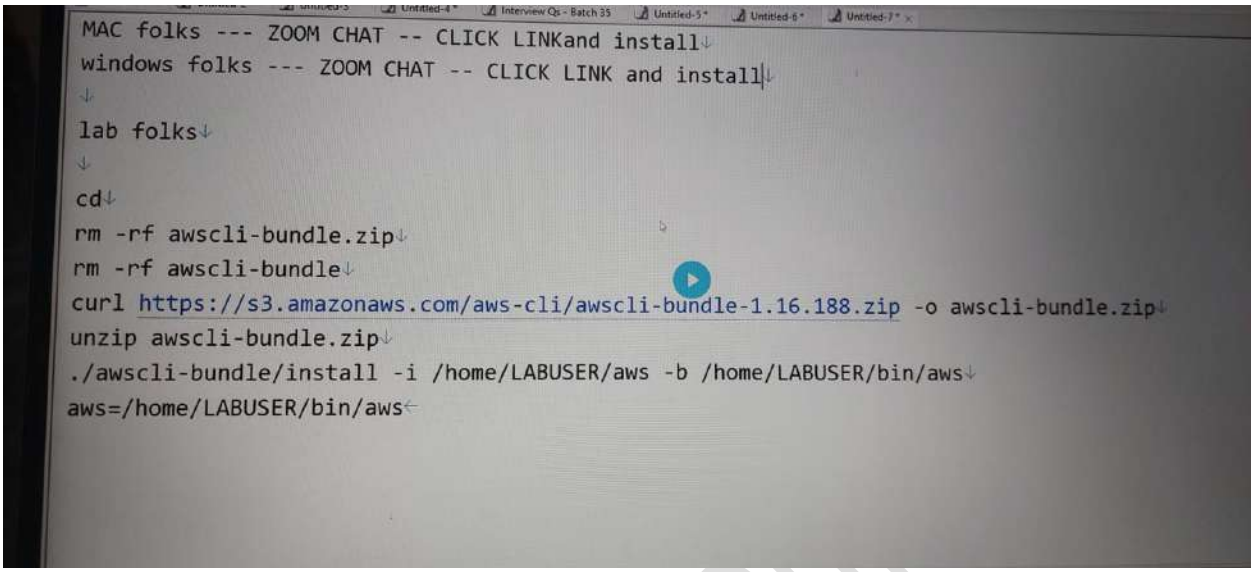
Project Description:



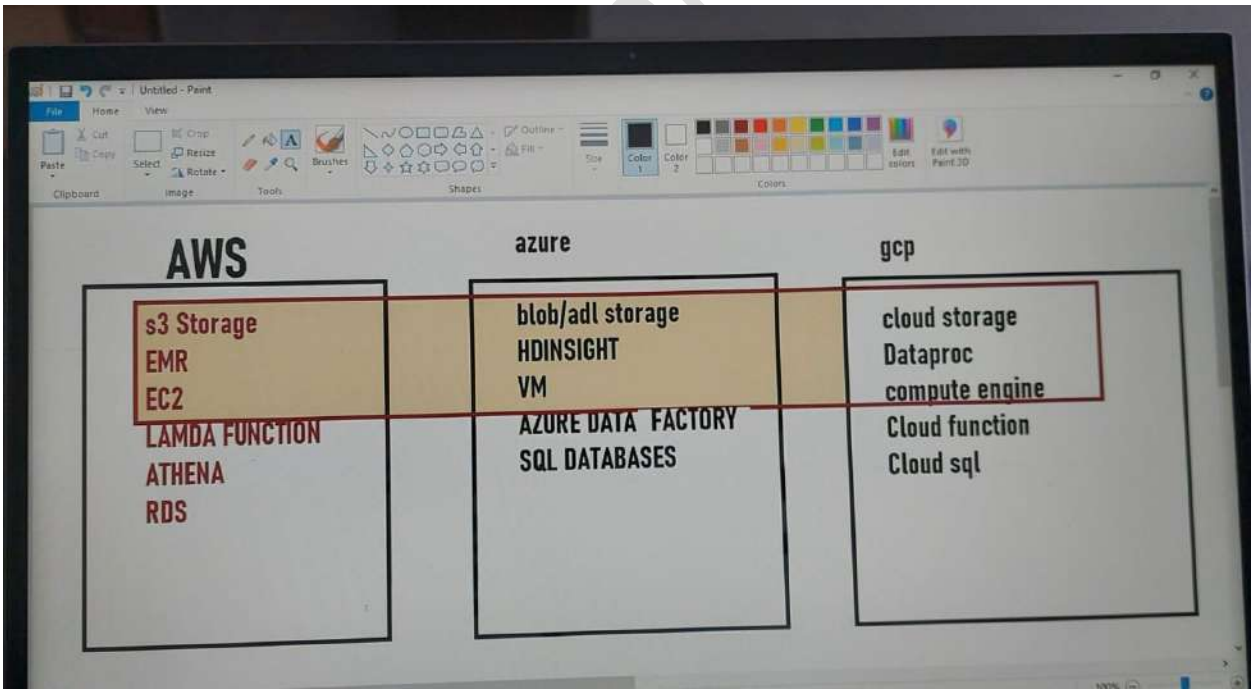
AWS Intro S3 Commands
Agenda Day 38

AWS Intro
Install Aws CLI on Windows and Mac – Click and Install

AWS Installation:



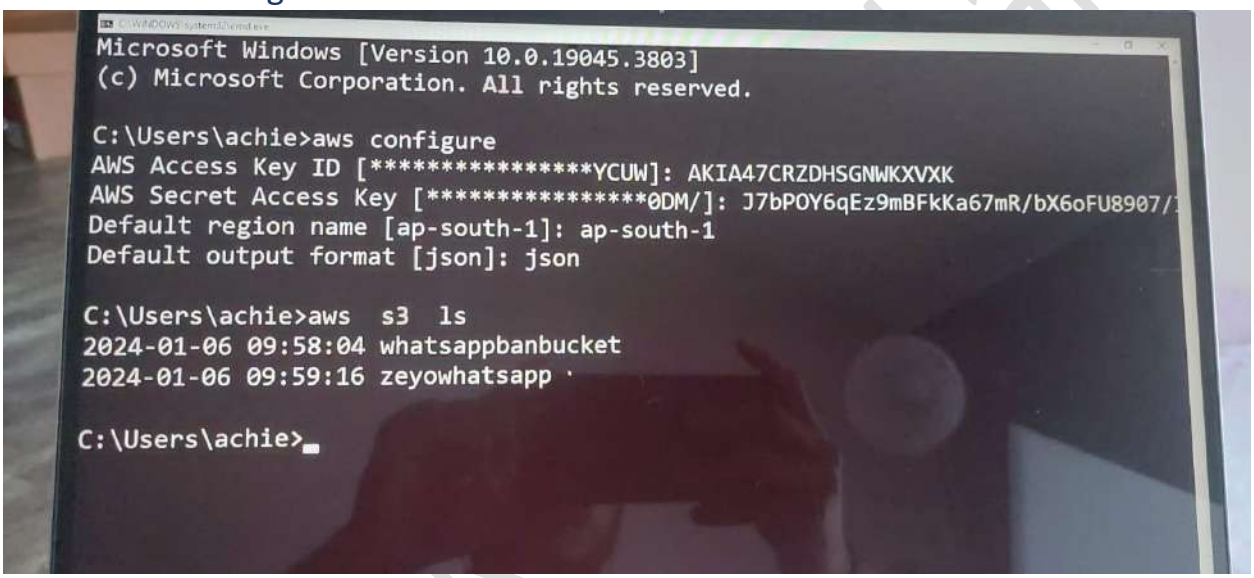
AWS services for Big Data:
S3 Storage, EMR, EC2 (Important), LAMBDA Function, Athena, RDA



S3 Hands-on

S3 – Simple storage service
Access Key and Secret to access AWS Services can be created from Profile → Security Credentials

Access the S3 from a command prompt:
1) Configure Access Key and Secret Key
Aws configure



AWS CLI Commands:

aws s3 ls → list Buckets
aws s3 mb → make bucket
aws s3 rb → remove bucket
aws s3 cp → copy
aws s3 mv → move
Aws s3 Sync → sync
aws s3 presign → URL generation
aws s3 website → for website hosting

Create Bucket:
aws s3 mb s3://<name_of_bucket>

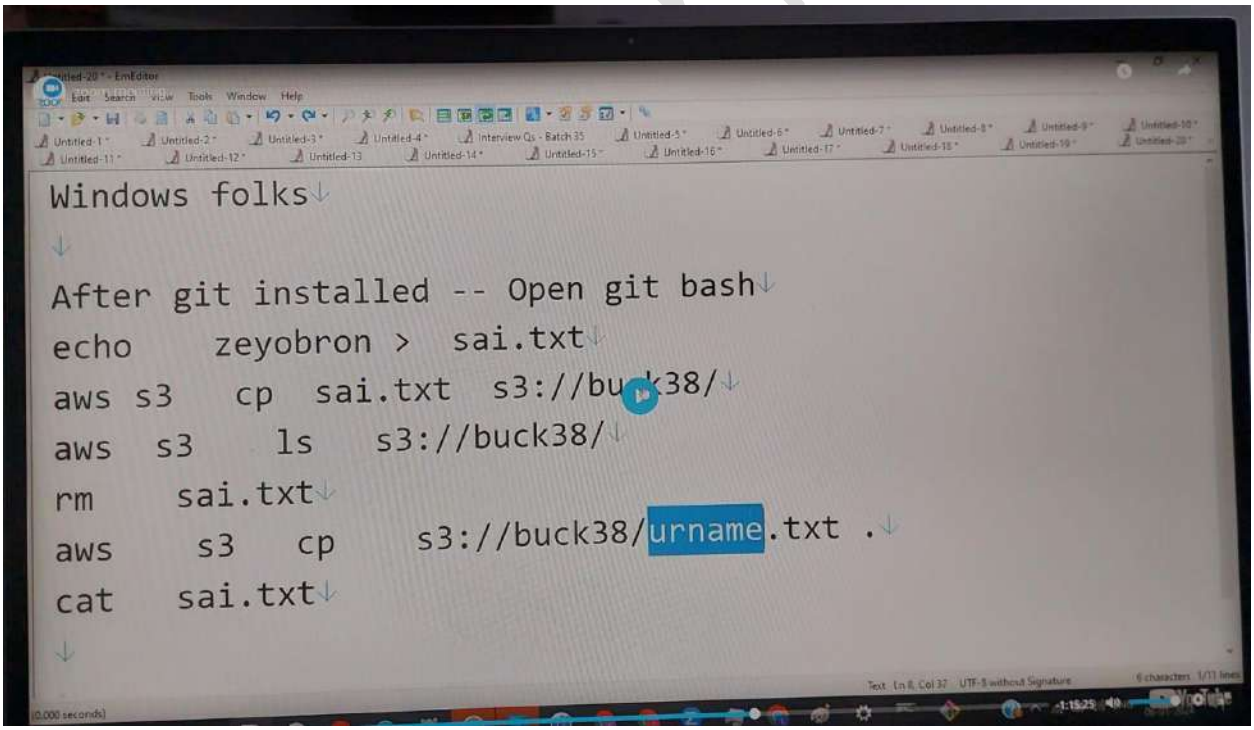
remove Bucket:
aws s3 rb s3://<name_of_bucket>

Copy local file to s3Bucket:
aws s3 cp <source_path> s3://<destination_bucket>

Remove file from Bucket:
aws s3 rm s3://<bucket_name>/<file_name>

List files in Buckets
aws s3 ls s3://<bucket_name>

task:



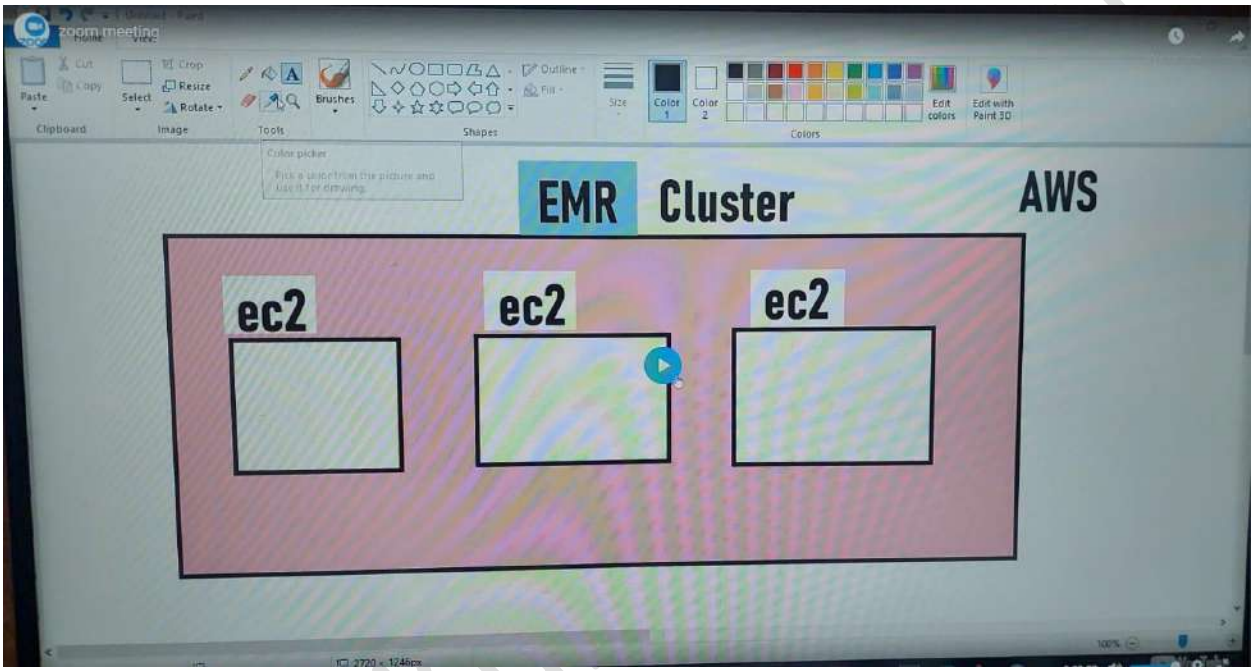
AWS s3 commands ec2 emr development

Agenda Day 39

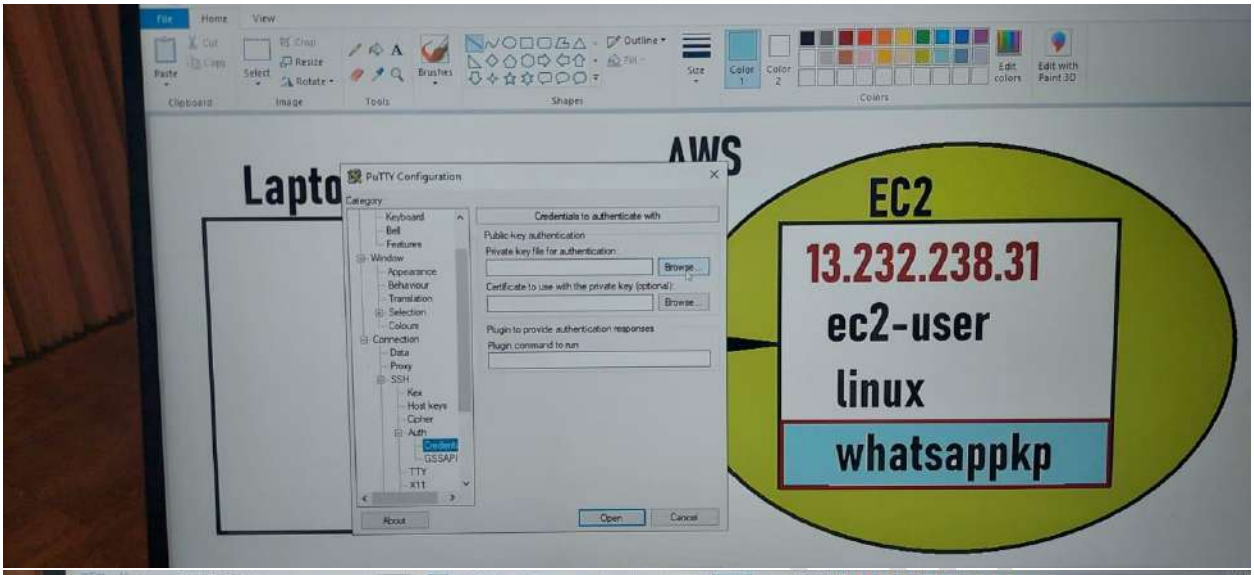
S3 Commands

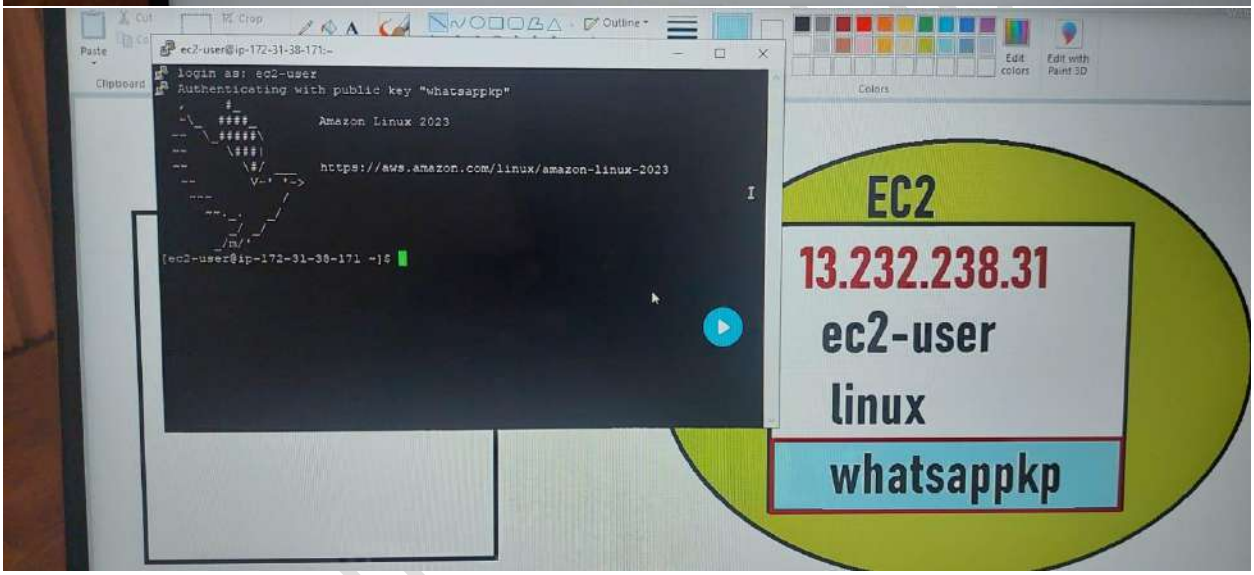
aws s3 mv <soure_path> <destination_Path> -- will move the file completely.

Ec2 Instance Creation



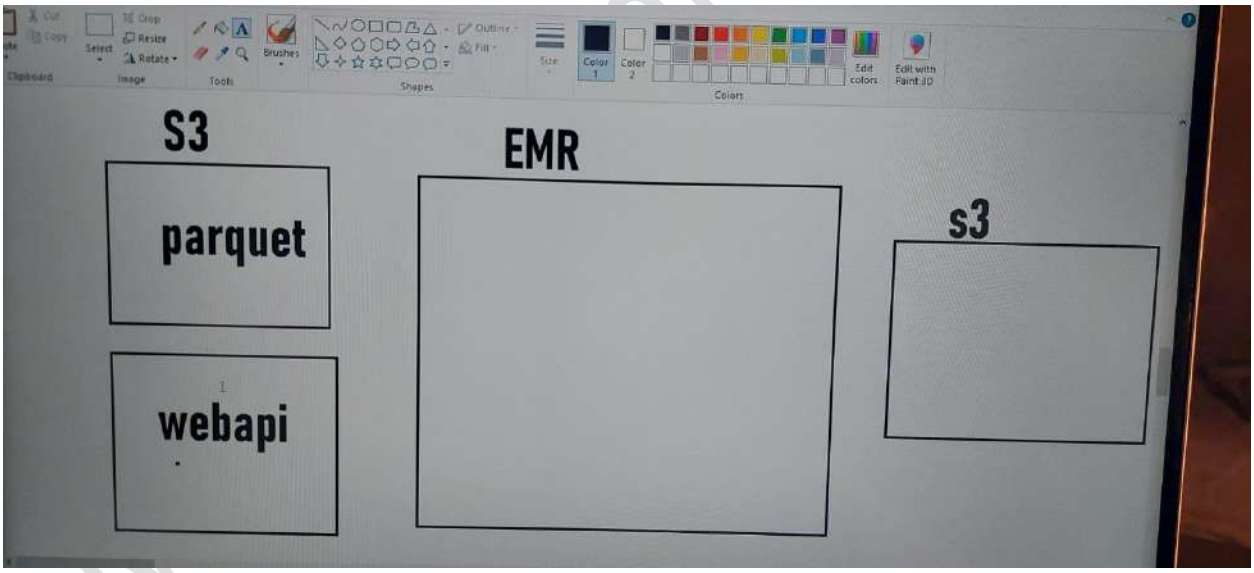
After Creating Ec2 Instances with Linux os, to login using putty, we need EC2 public IP, username (ec2-user always) and the key-pair (to be uploaded as shown in below screenshot)





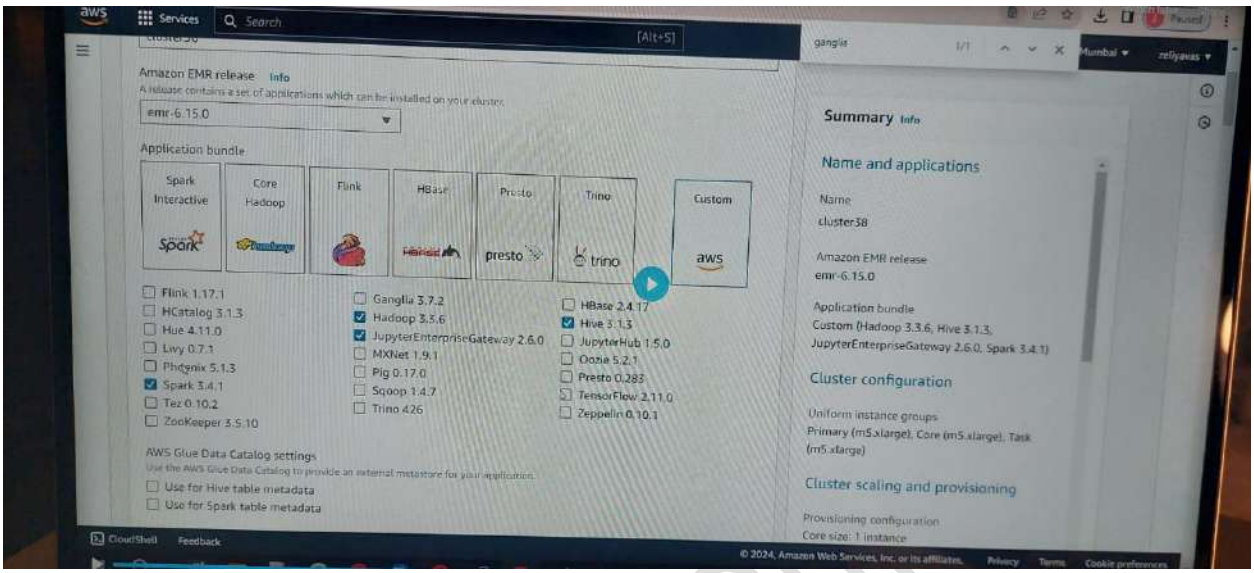
Download putty:
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html


WhatsApp
Install the gcloud CLI | Google
Download PuTTY: latest release
chiark.greenend.org.uk/~sgtatham/putty/latest.html
Download PuTTY: latest release (0.80)
Home | FAQ | Feedback | Licence | Updates | Mirrors | Keys | Links | Team
Download: Stable | Snapshot | Docs | Changes | Wishlist
This page contains download links for the latest released version of PuTTY. Currently this is 0.80, released on 2023-12-18.
When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternatively, here is a permanent link to the 0.80 release.
Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date version of the code available. If you have a problem with this release, then it might be worth trying out the development snapshots, to see if the problem has already been fixed in those versions.
Package files
You probably want one of these. They include versions of all the PuTTY utilities (except the new and slightly experimental Windows pterm).
(Not sure whether you want the 32-bit or the 64-bit version? Read the FAQ entry.)
We also publish the latest PuTTY installers for all Windows architectures as a free-of-charge download at the Microsoft Store; they usually take a few days to appear there after we release them.
MSI ('Windows Installer')
64-bit x86: putty-64bit-0.80-installer.msi (signature)
64-bit Arm: putty-arm64-0.80-installer.msi (signature)
32-bit x86: putty-0.80-installer.msi (signature)
Unix source archive
.tar.gz: putty-0.80.tar.gz (signature)
Alternative binary files

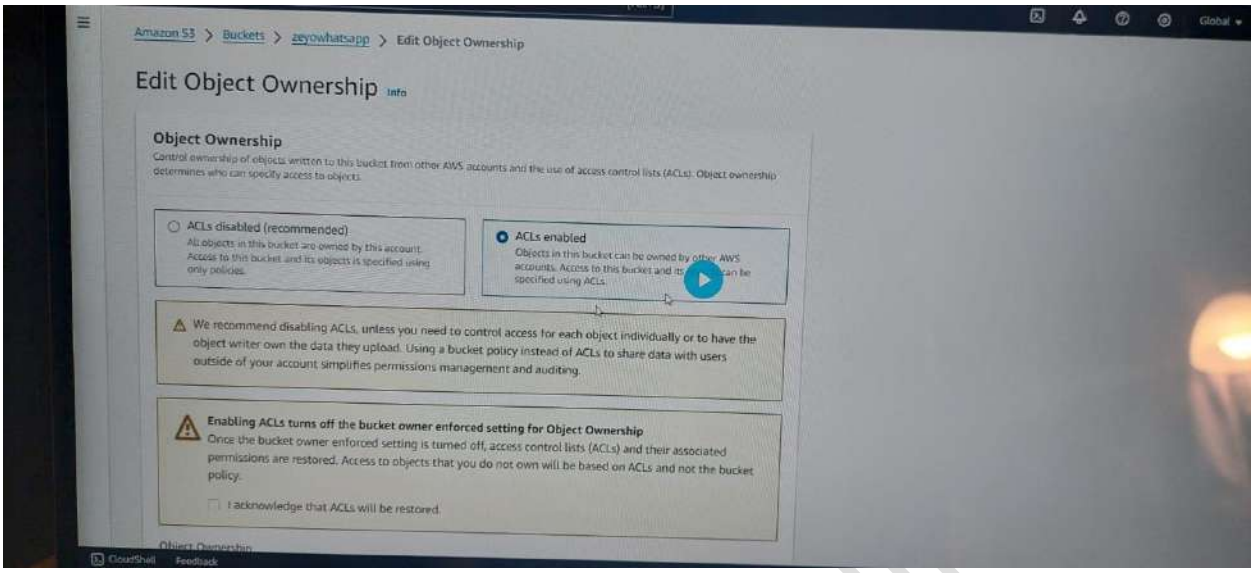

S3
parquet
webapi
EMR
s3

EMR Service in Amazon:

Ganglia also to be selected

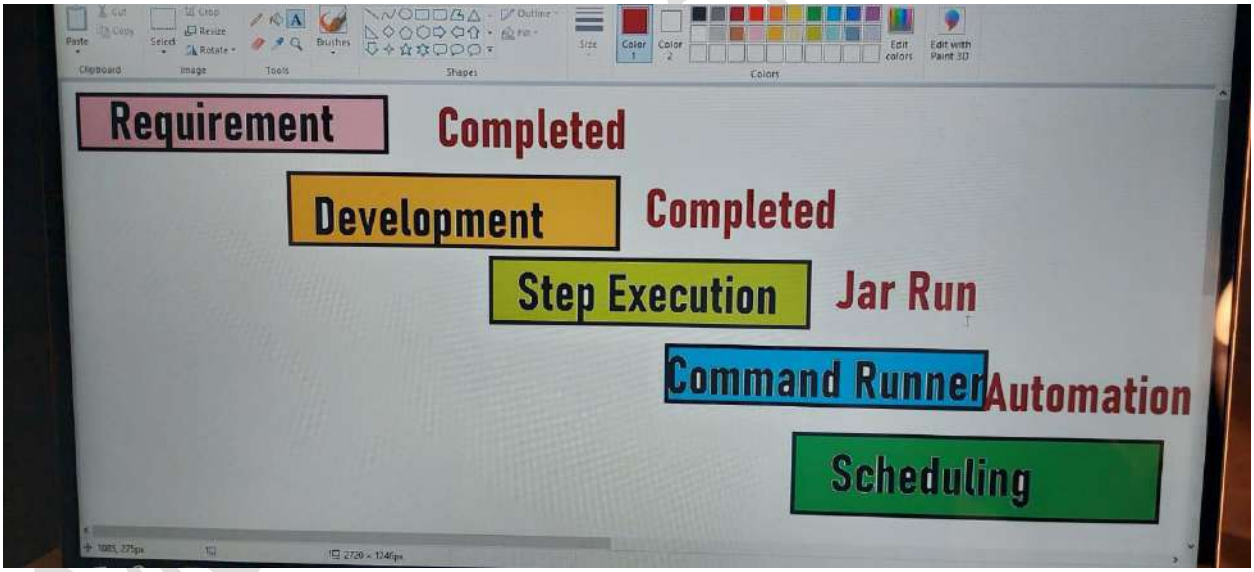


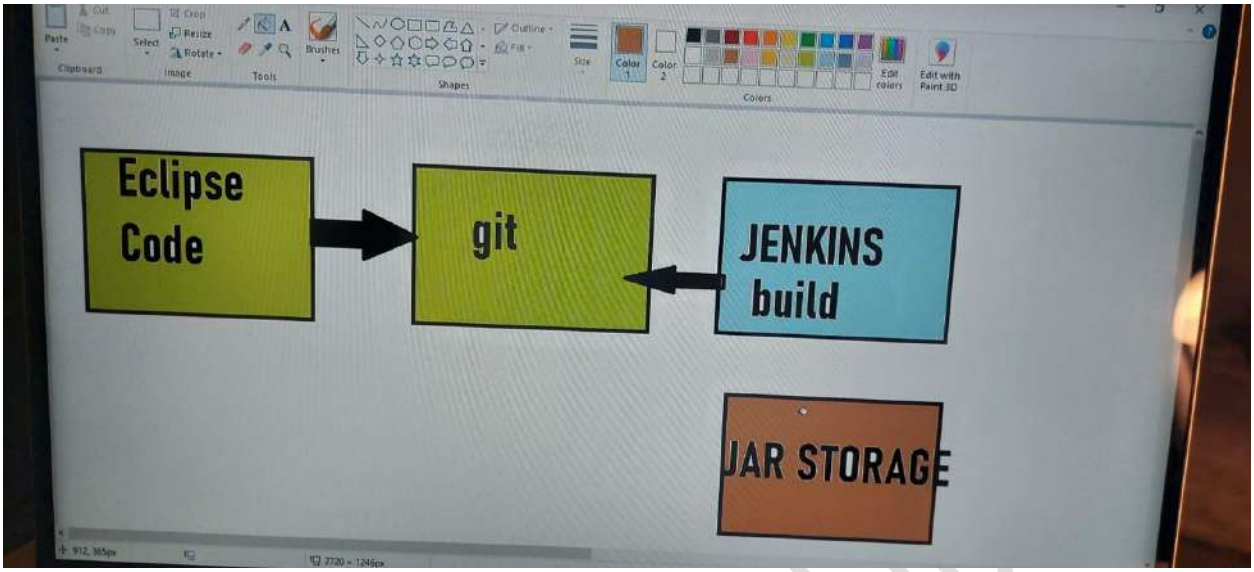
Once the cluster is created take the public node public DNS(Ec2 DNS name)
go to putty, Give the Hostname as public node public DNS and provide key-pair authentication and connect.
Always add SSH in inbound rules. User name for AWS EMR is hadoop.
Once we connected the EMR from putty, open spark-shell and execute the Spark code.
To permit files in the Bucket , go to the bucket → permissions →Accss Control List (click bucket owner enforced) →Select ACLs enabled , Check I Acknowledge that ACLs will be restored, select Object Writer and save changes



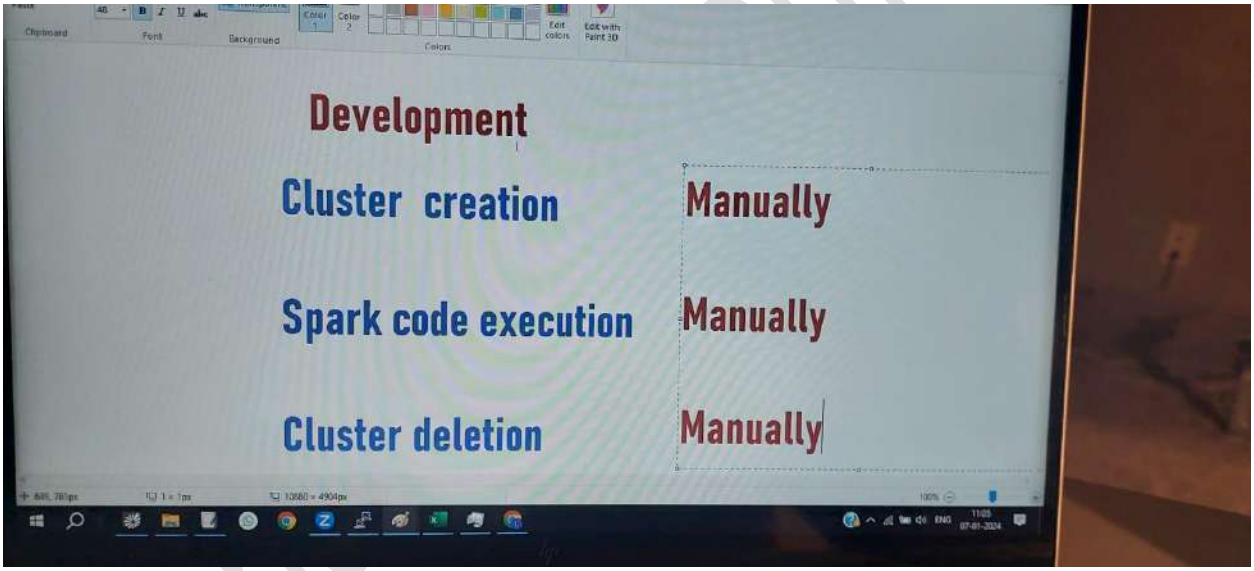
Also select the folder and click Actions → Make public using ACL

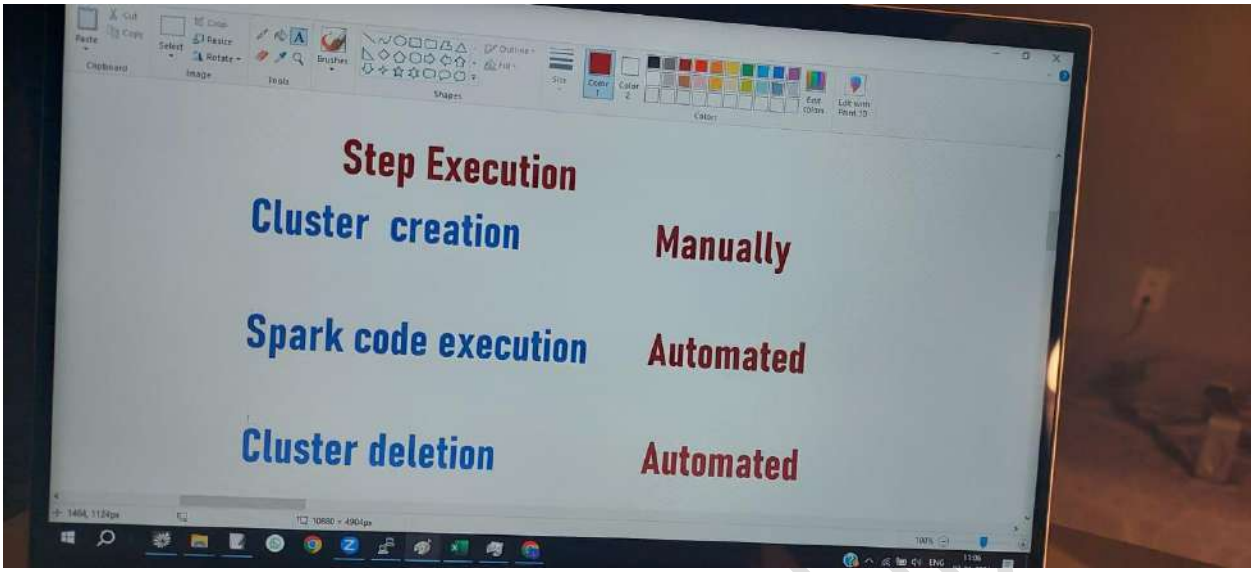
EMR development

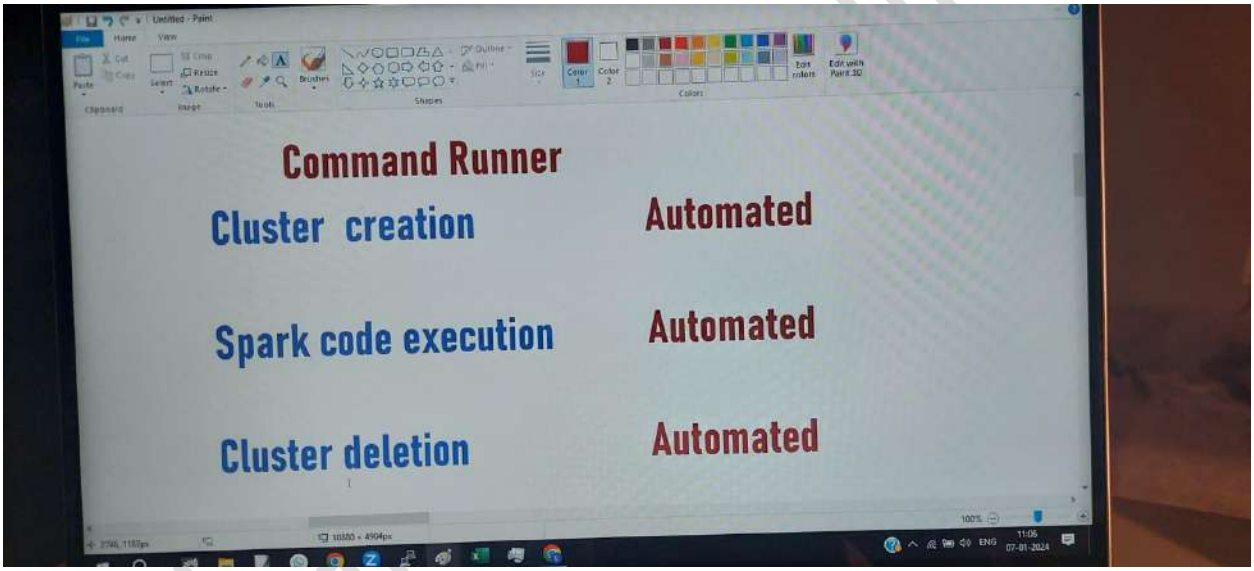




Place the jar in S3.



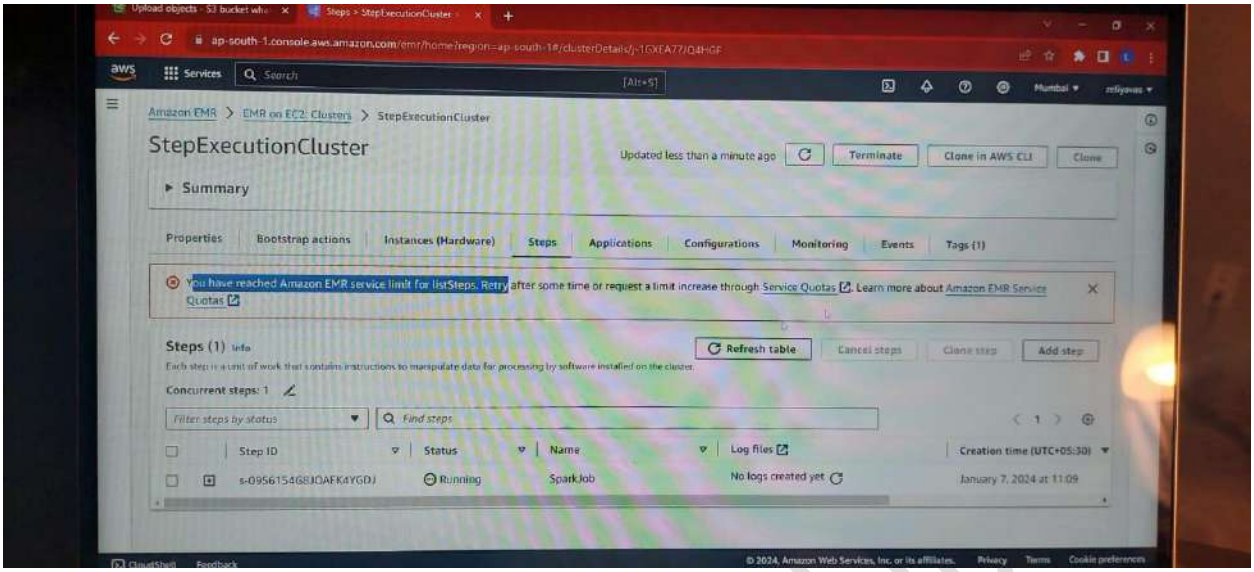




Step Execution:

1) Create a EMR Cluster
2) Under Step → Add Step → Spark Application → Give a name → In application location give the path of the jar in S3 → in Spark-Submit options (--class pack.obj) → Sace
3) We can also choose cluster termination
4) Create the cluster.

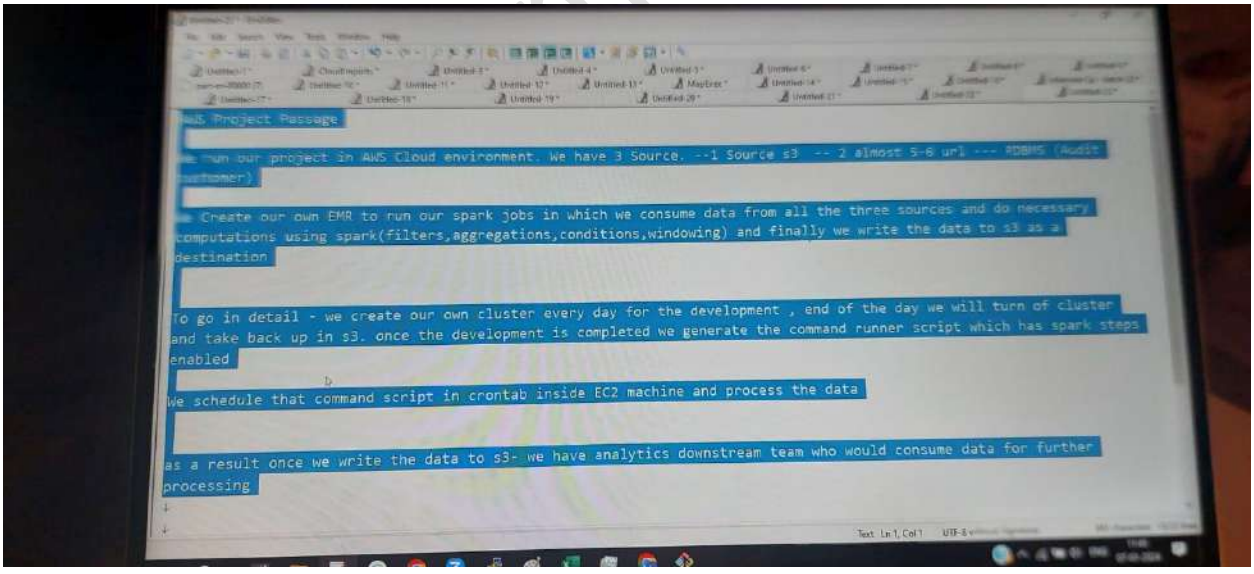
Once the cluster is created, the spark job bundled in the jar file will get executed. to see the status of the spark job(step), open the cluster, click on the steps option.



Command Runner Script:

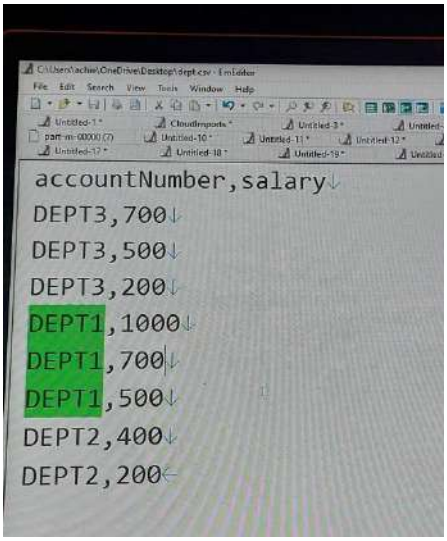
Click on “Clone in AWS CLI” (From the above screenshot) to get the command line code

AWS Project Passage:



Task:

Second Highest Salary – For Each department



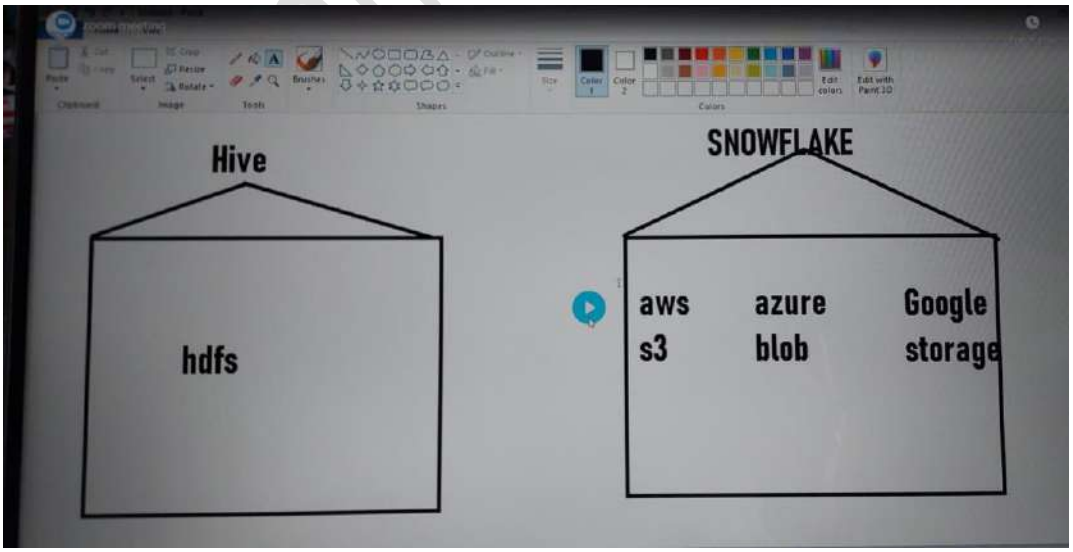
EMR Deployment

EMR Project deployment Command Runner

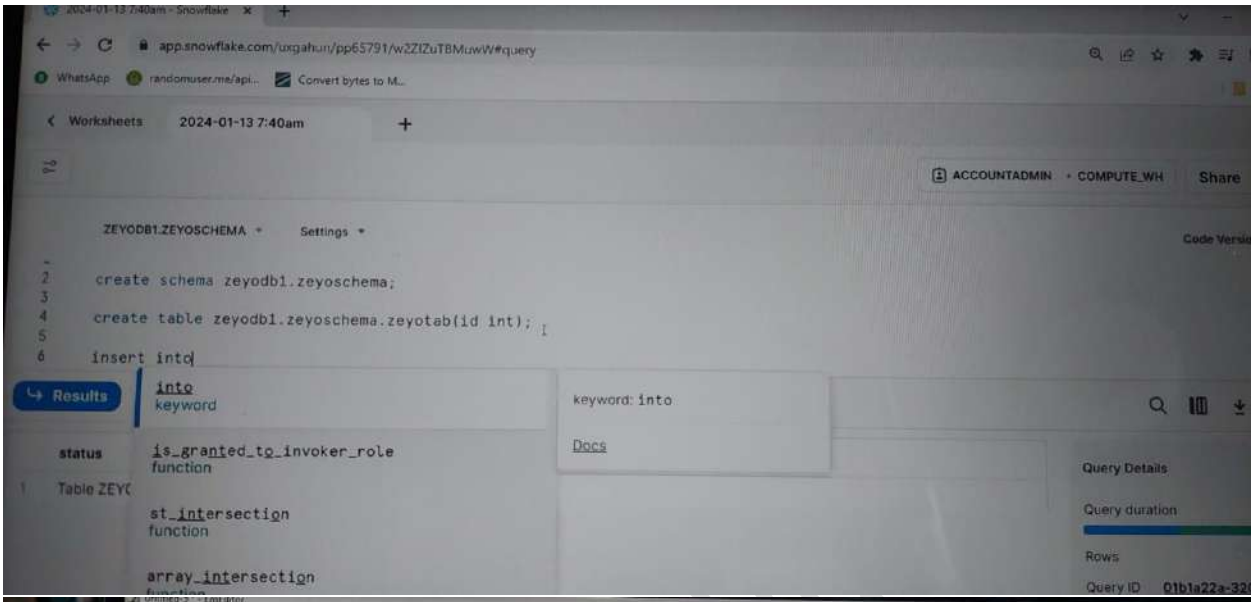
Agenda Day 40

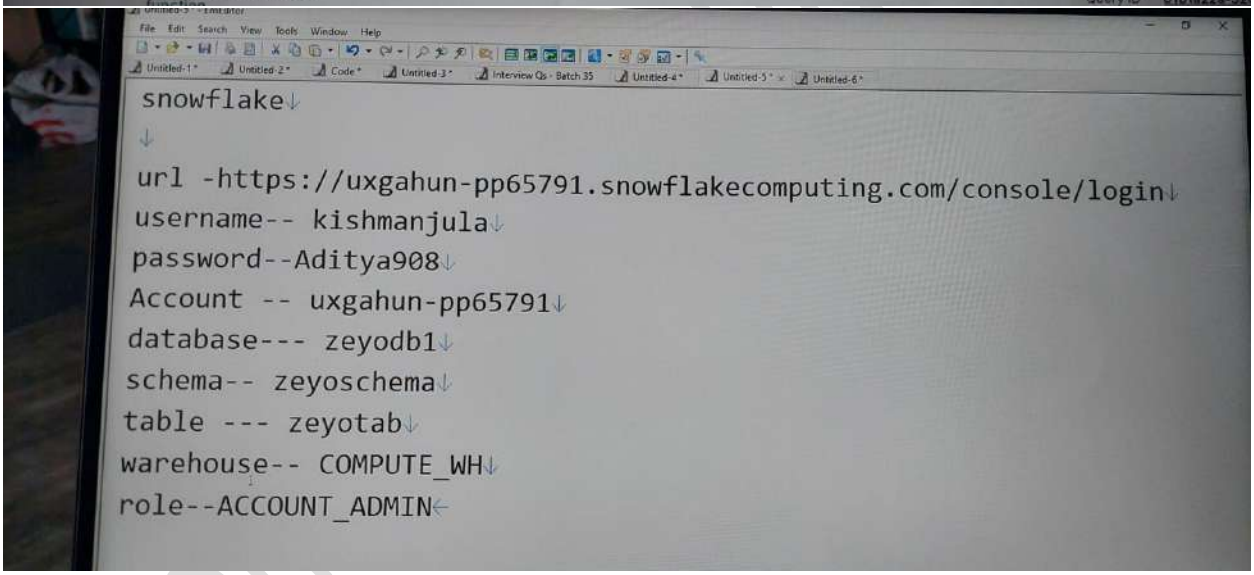
EC2 Instance Type – M,R(Memory Optimized),C Series → Explore on my own

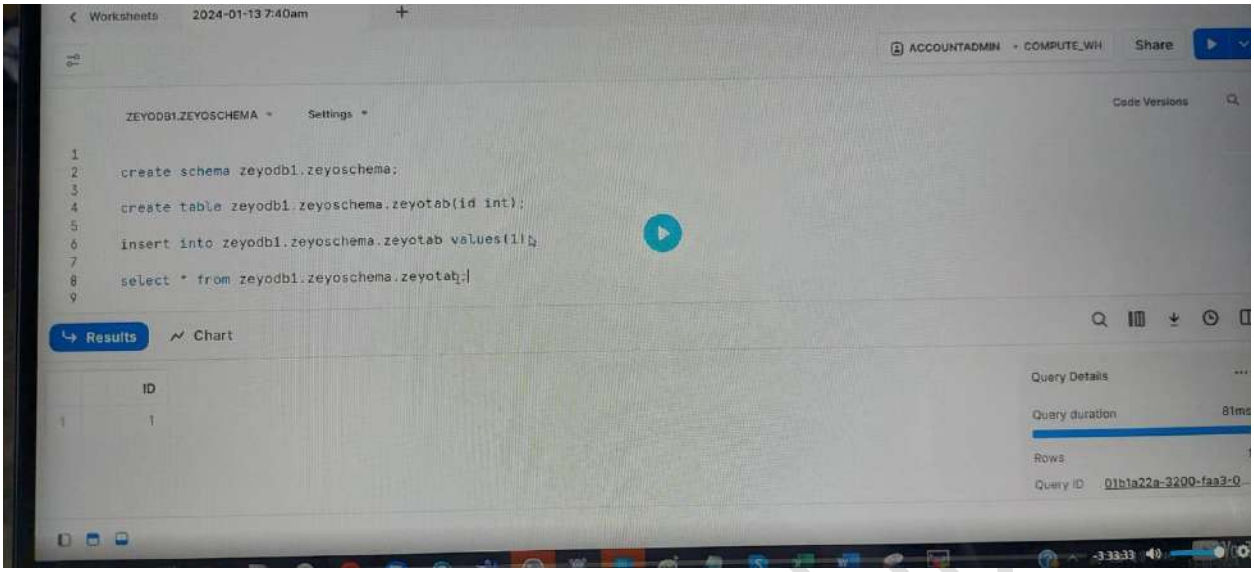
Snowflake – datawarehouse in the cloud environment



Important parameters that we have to understand in snowflake:
Url, username, password, Account, database, schema, table, warehouse, roles





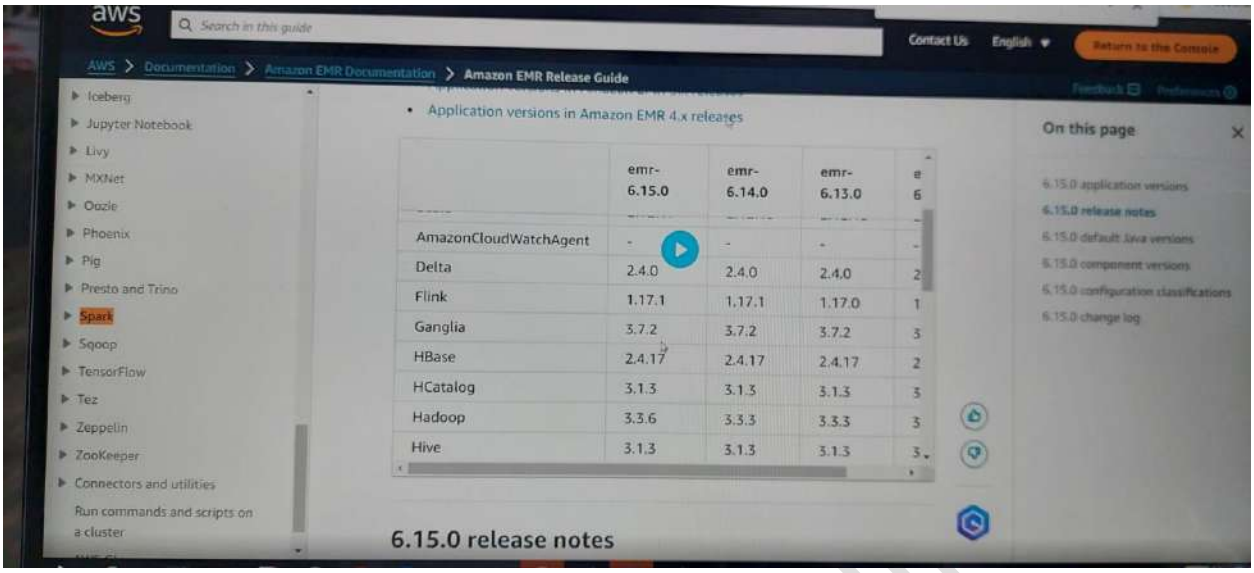


Scenario:

Read Project Parquet
Read Webapi data
Process it using spark
write it to snowflake table

Requirement:
Read the data from local and WebAPI, process it on EMR and write it to snowflake table.

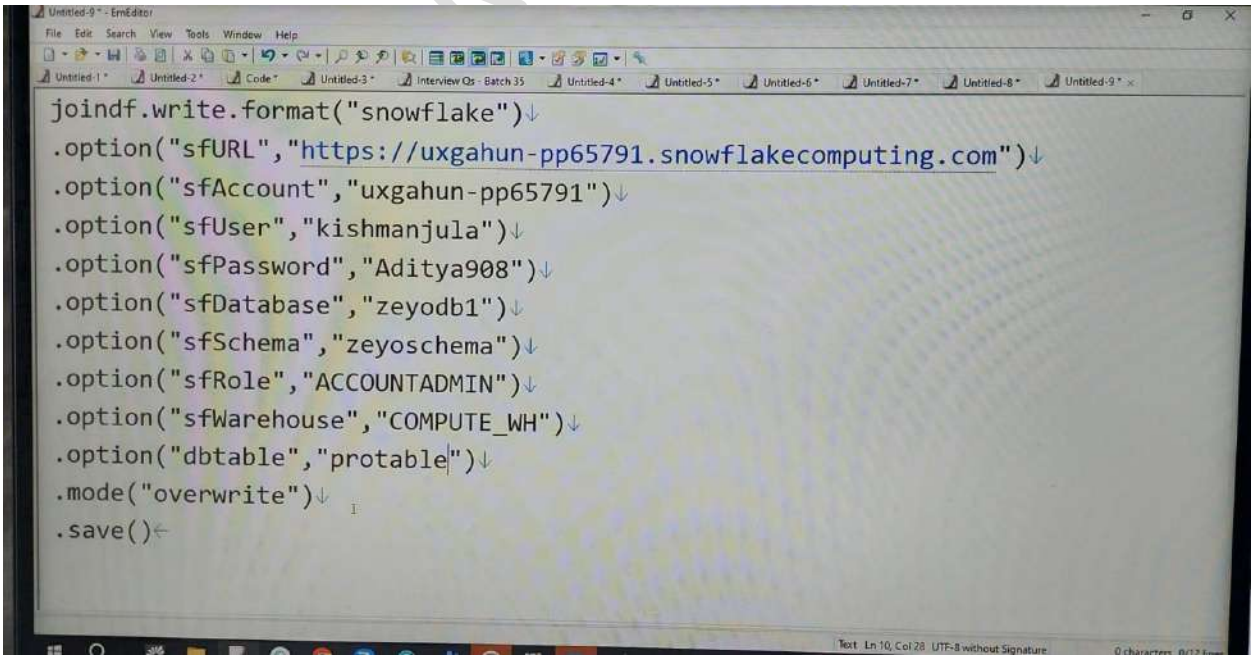
Steps:
1) Create an EMR cluster in AWS
2) Connect to this cluster using Putty
3)Open Spark-shell and snowflake connector.
to get the versions of software on the EMR created, we need to get the EMR version and then web search to get the relevant versions.

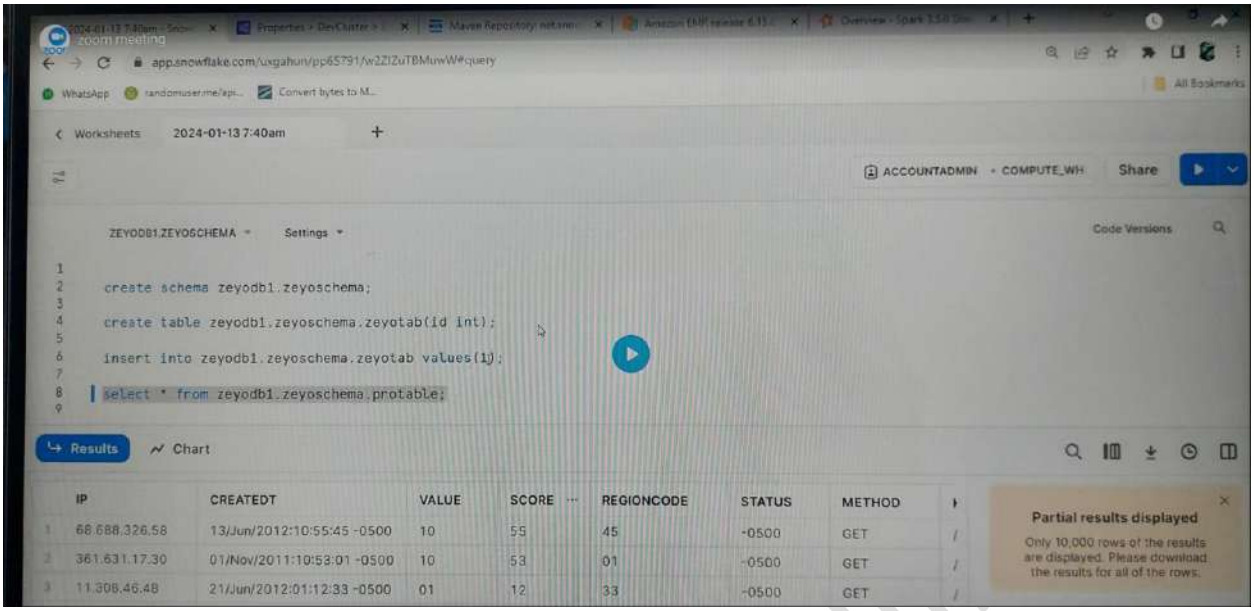


Command to open the spark-shell:
spark-shell --packages net.snowflake:spark-snowflake_2.12:2.12.0-spark_3.4
<package>:<artifact>:<version>

Confirm spark-shell installation confirmation
Just do import net.snowflake.

Write to snowflake -DB:

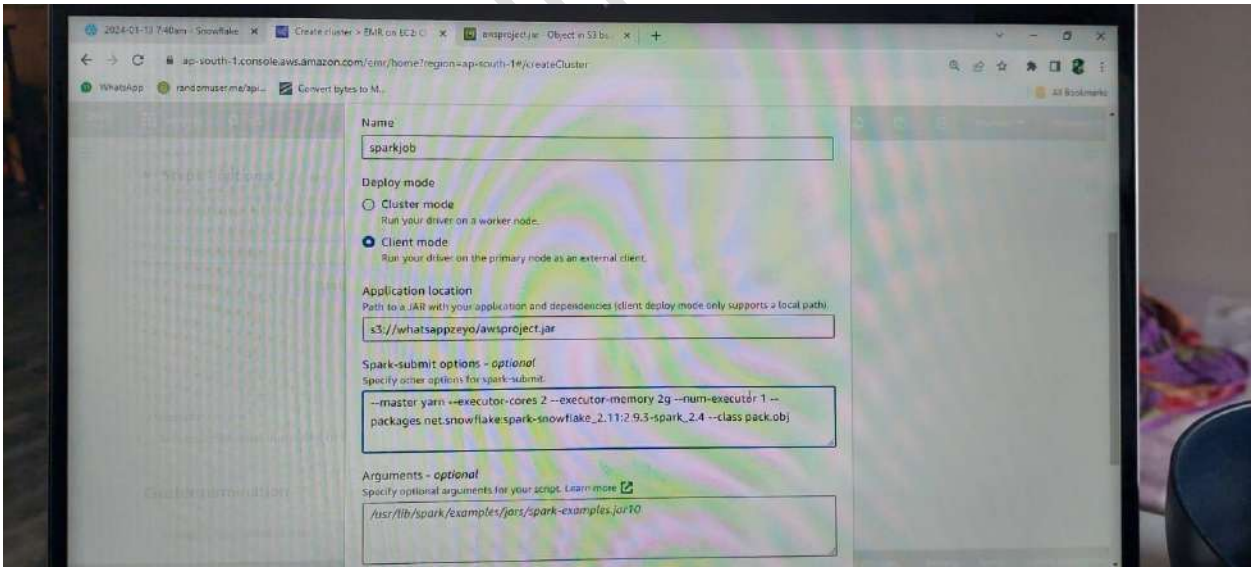


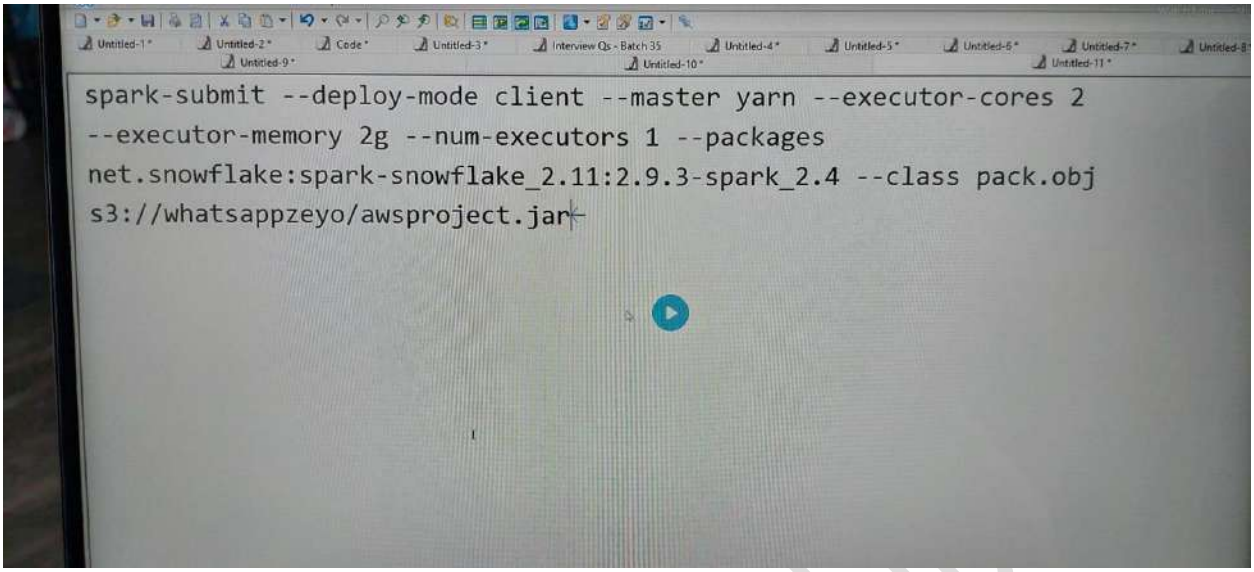


Development completed – So terminate the cluster.

Now Export the code as JAR, and upload that to S3 bucket and make itas public (By clicking on actions in S3 Bucket by selecting the file).

Now, Create a Step execution cluster. Specify Spark-submit options





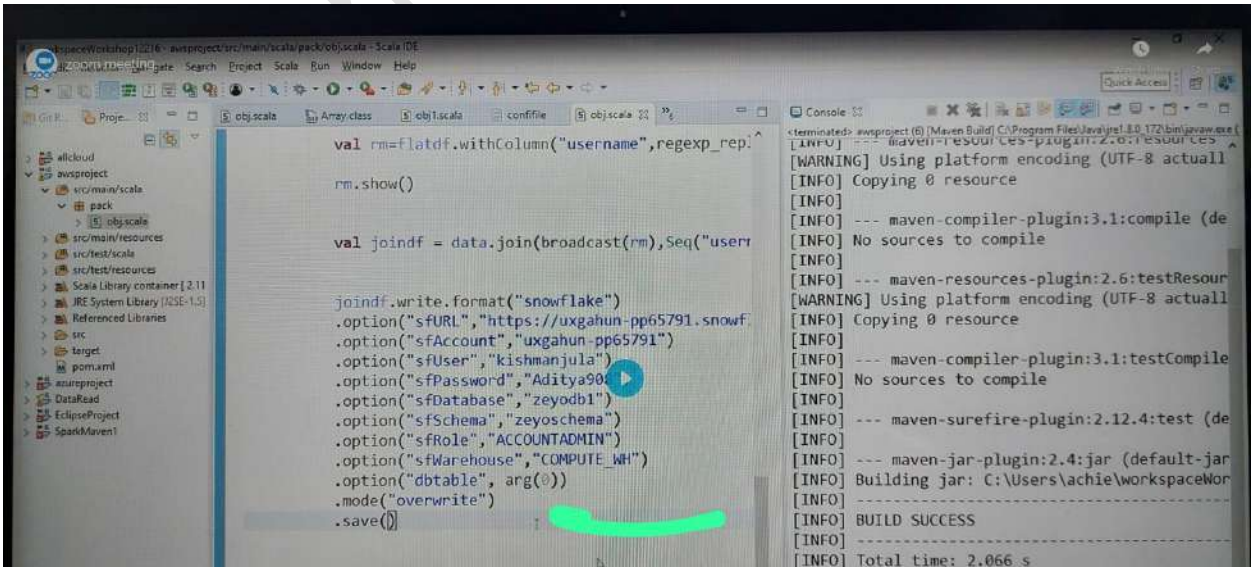
If error, we can add another step and it gets executed.

Ganglia is to view Cluster overview.
Cluster phases once started → Starting, Running, Waiting (if auto-terminated is selected it would be terminating)

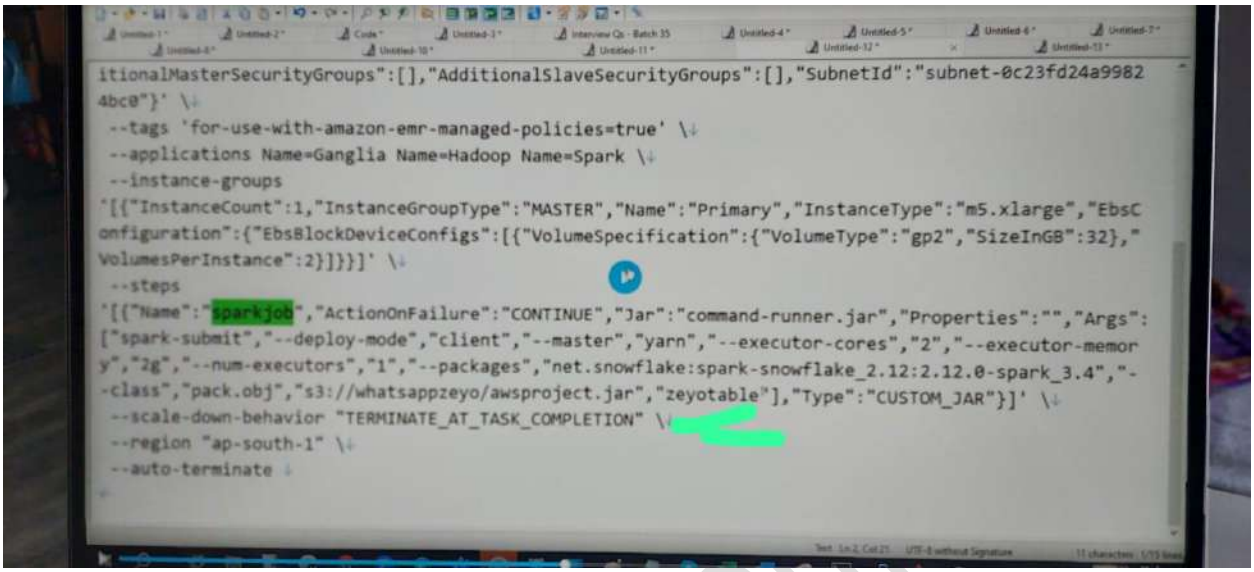
Now, as next step, we could trigger the Jobs using AWS CLI COMMAND

Practice:

Modify the code to accept args from the main method parameters



To pass these args, we can add the parameter in the command line command after jar file S3 Path as below.



When we trigger this command the clusters will start, To list the clusters, below is the command **aws emr list-clusters**

To get the cluster state: aws emr describe-cluster –cluster-id <Cluster_ID> | grep ‘State’

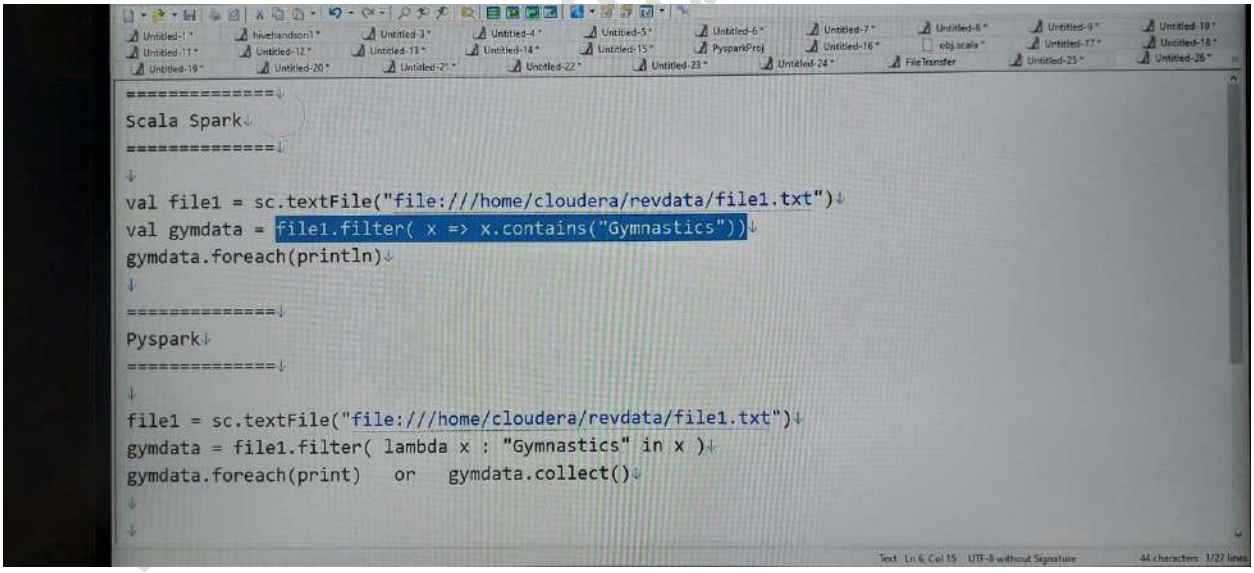
Spark EMR Deployment pyspark into Handson rdd
Agenda Day 41
Pyspark Intro(Why)

Spark triggered using 4 languages – Scala, Python, Java, R.

Project Requirements decide which language to be opted, for based on the libraries it supports. For example, urlopen is available to read both rest/soup url in python , where as reading soup url is not possible in Scala (in 2021 around)

Requirements:
Install Python
Download Winscp → Winscp is used to transfer files to a remote server.

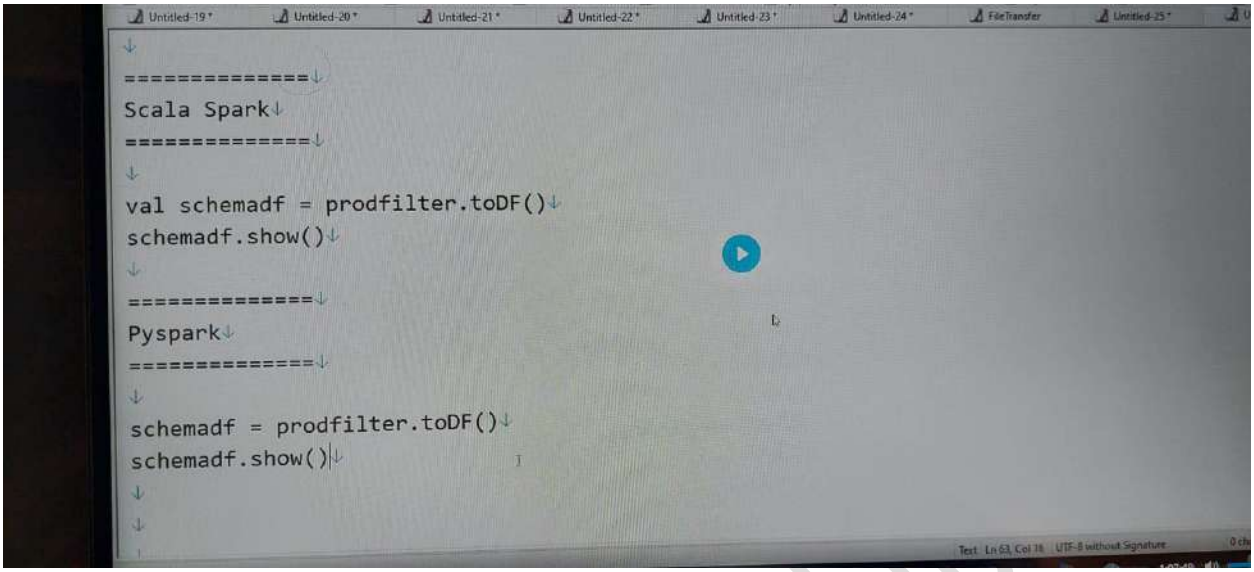
Pyspark Handson



```
Scala Spark
==========

case class
schema(txnno:String,txndate:String,custno:String,amount:String,category:String,product:String,city:
String,state:String,spendby:String)

val mapsplit = gymdata.map( x => x.split(","))

val schemardd = mapsplit.map( x => schema(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))

val prodfilter = schemardd.filter( x => x.product.contains("Gymnastics"))

prodfilter.foreach(println)
```

Python Code:

```
Pyspark
==========

from collections import namedtuple

schema =
namedtuple("schema",["txnno","txndate","custno","amount","category","product","city","state","spend
by"])

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]))

prodfilter = schemardd.filter(lambda x : "Gymnastics" in  x.product)

prodfilter.foreach(print)
```



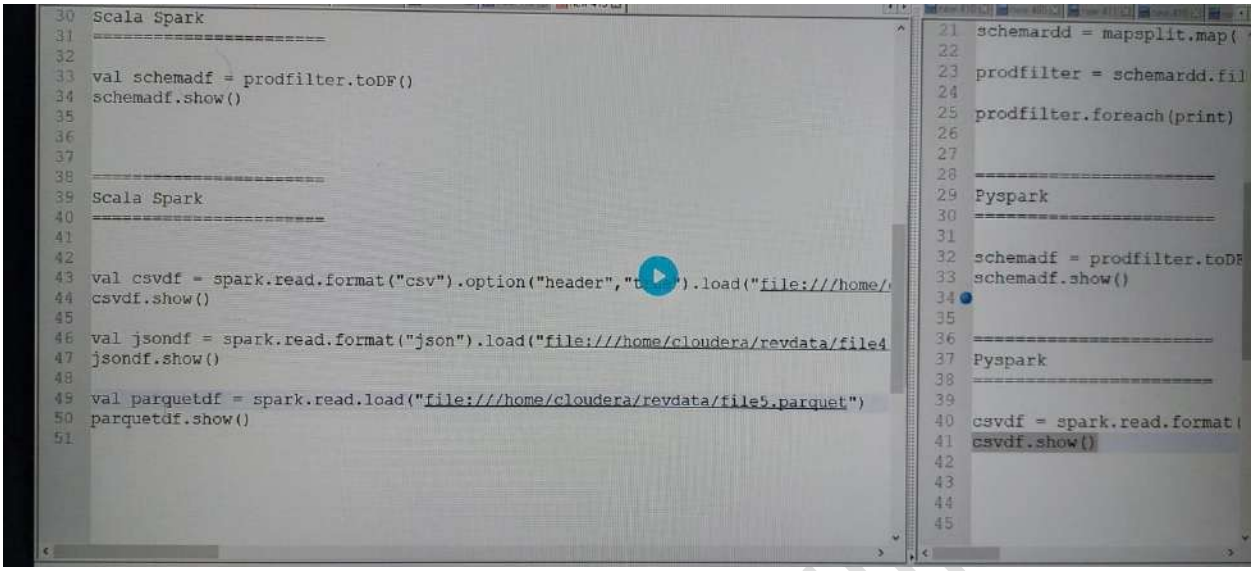
Pyspark Development EMR Development
Agenda Day 42

Scenarios Codes:
Collecting the Group results as List (Scala):
Collect.groupBy("id").agg(collect_list("subject").as("subject"))
pivot – is used to convert rows to columns

sub.withColumn("name",expr("explode(split)"))

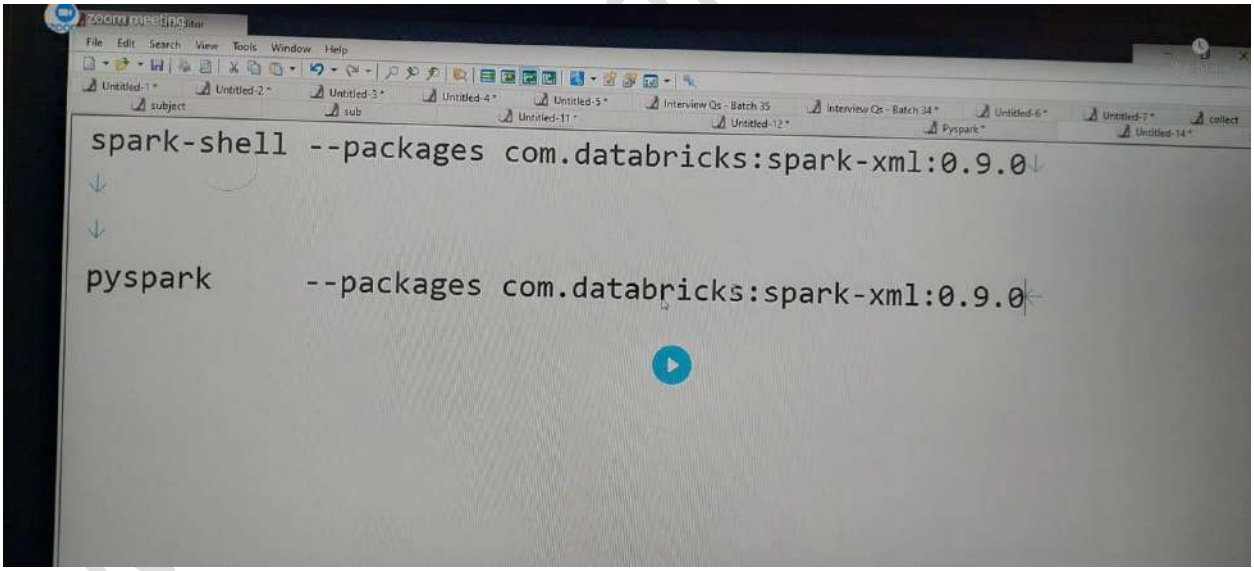
I have 1000 files in HDFS, I want to pull it in spark and have my output as a dataframe like FileName | Contents of File

Spark vs Pyspark



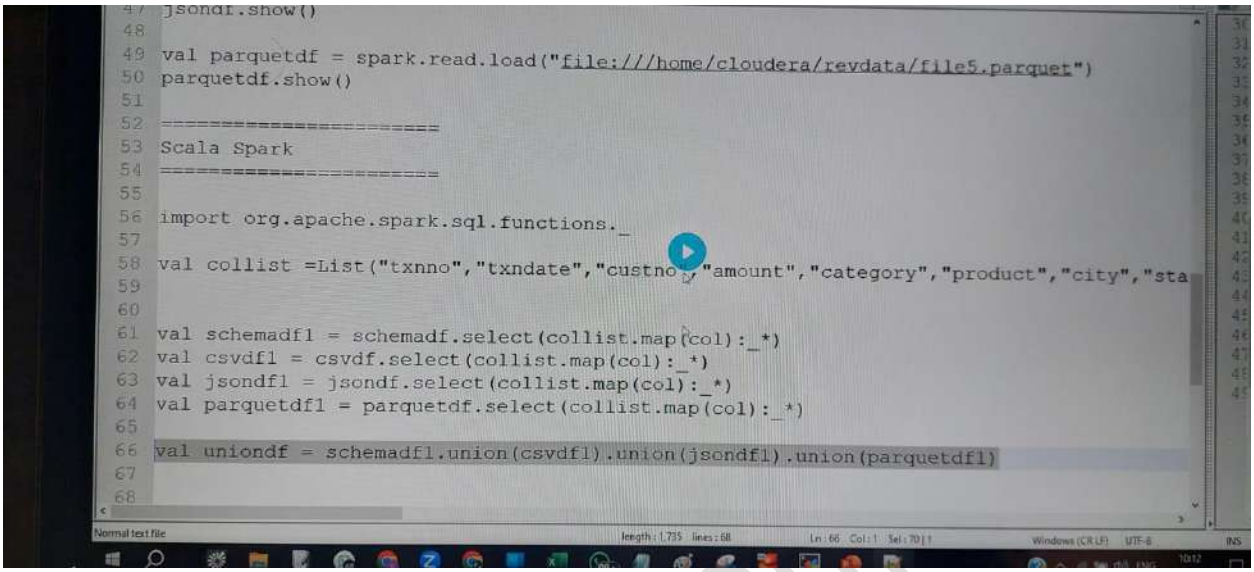
Only remove val in the above screen for pyspark.

for reading XML:

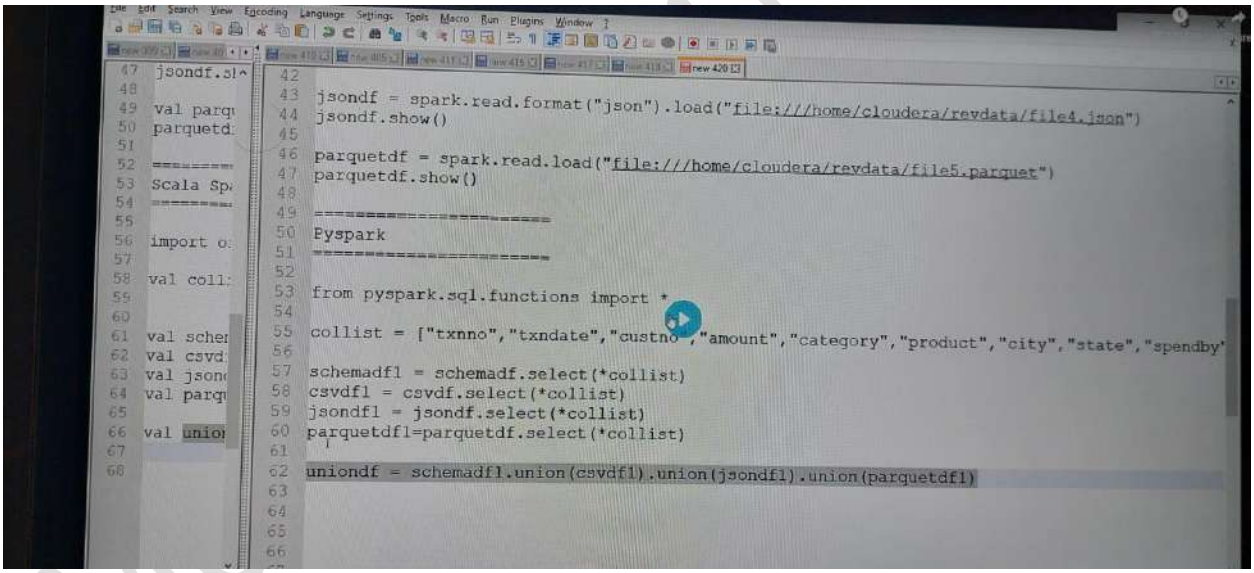


Mapping Columns for data:

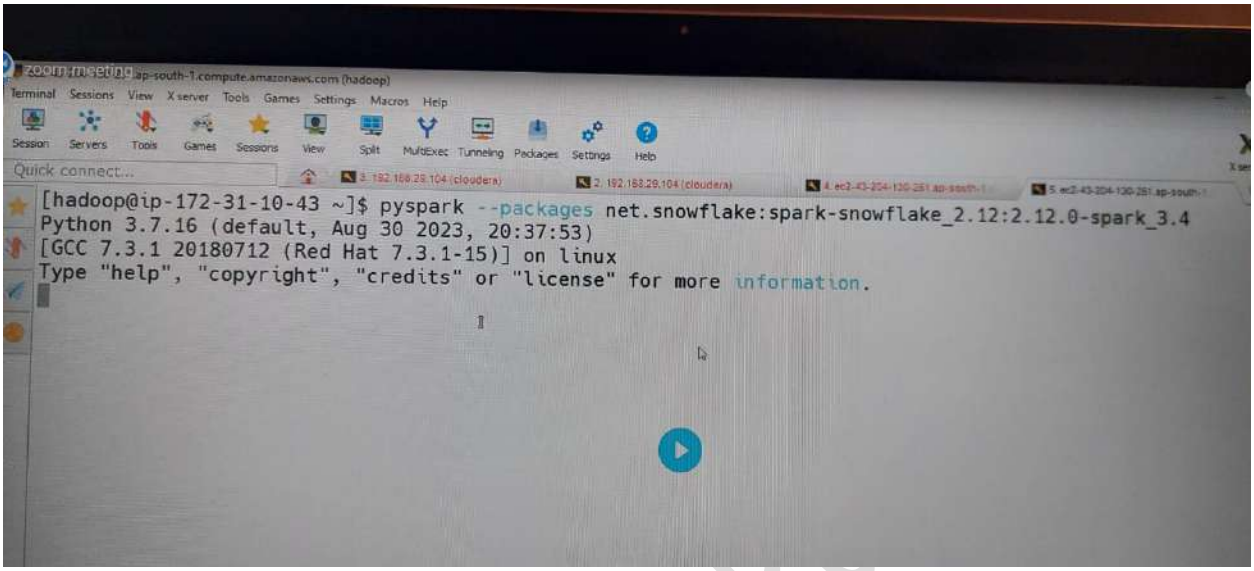
Spark



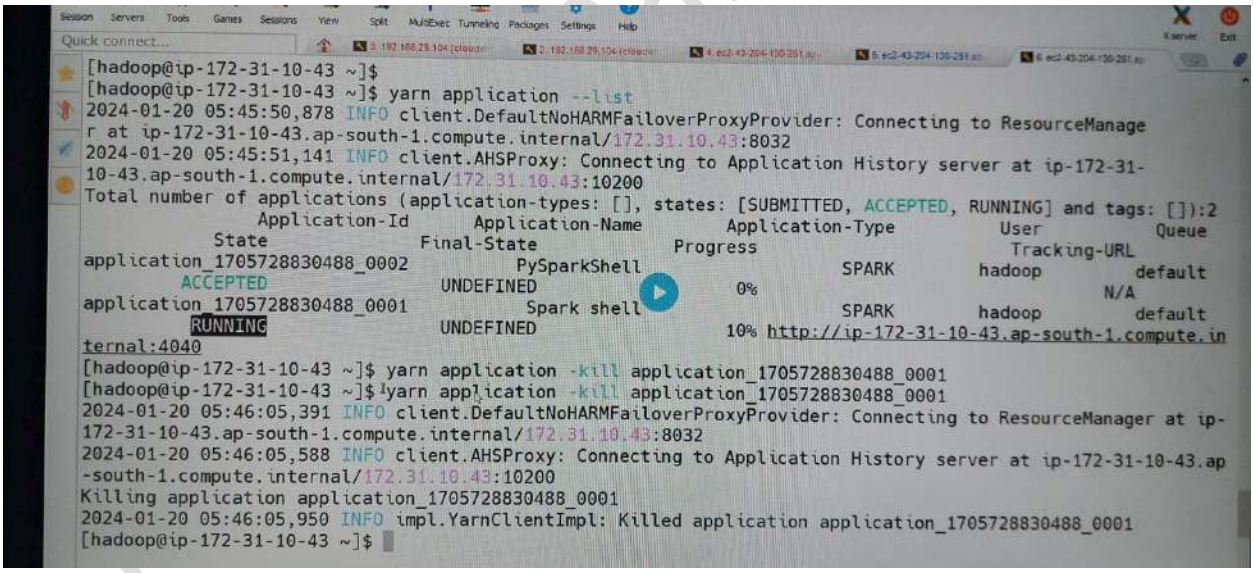
Pyspark



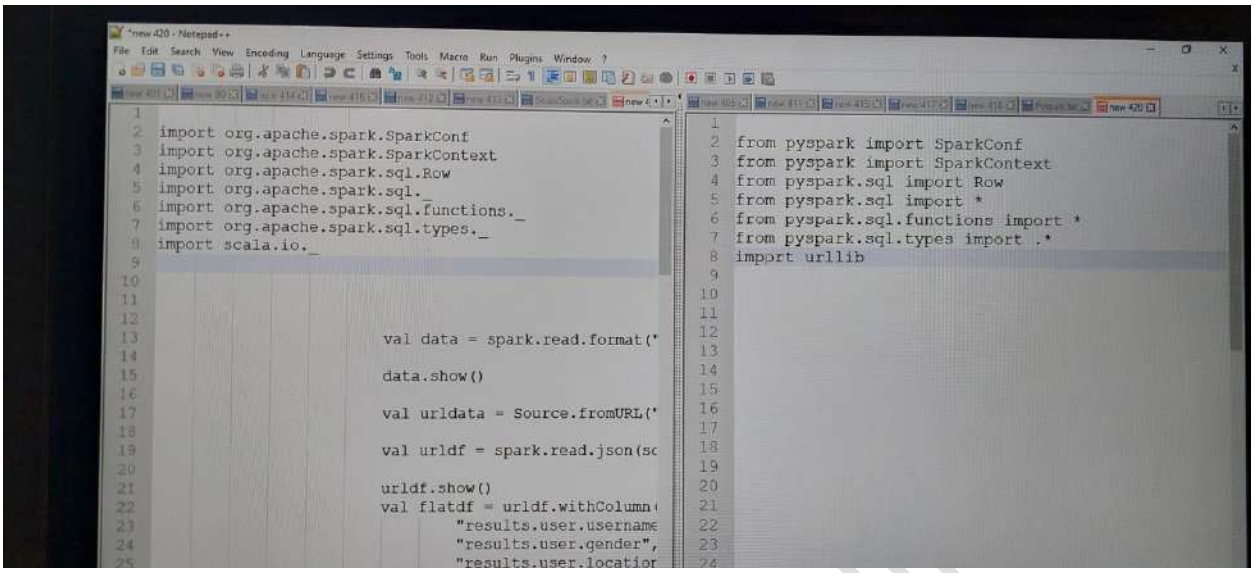
Pyspark development on AWS EMR:
Launching pyspark in EMR



yarn application --list → gives the list of shells opened
yarn application -kill <application_id>



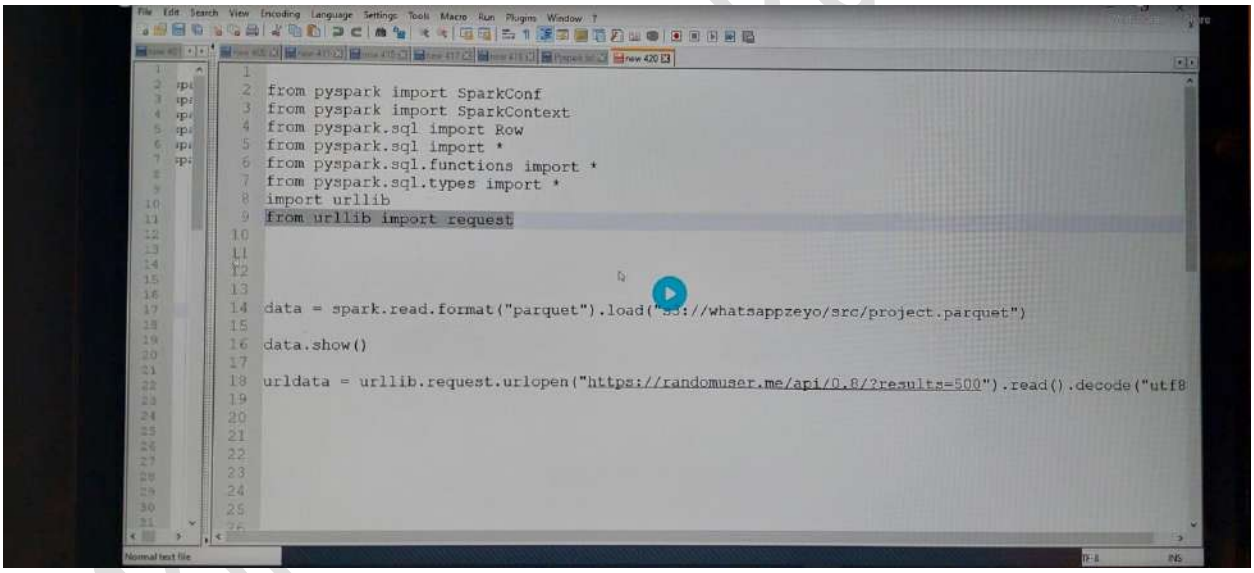
Imports:



```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.Row
import org.apache.spark.sql._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import scala.io._
```

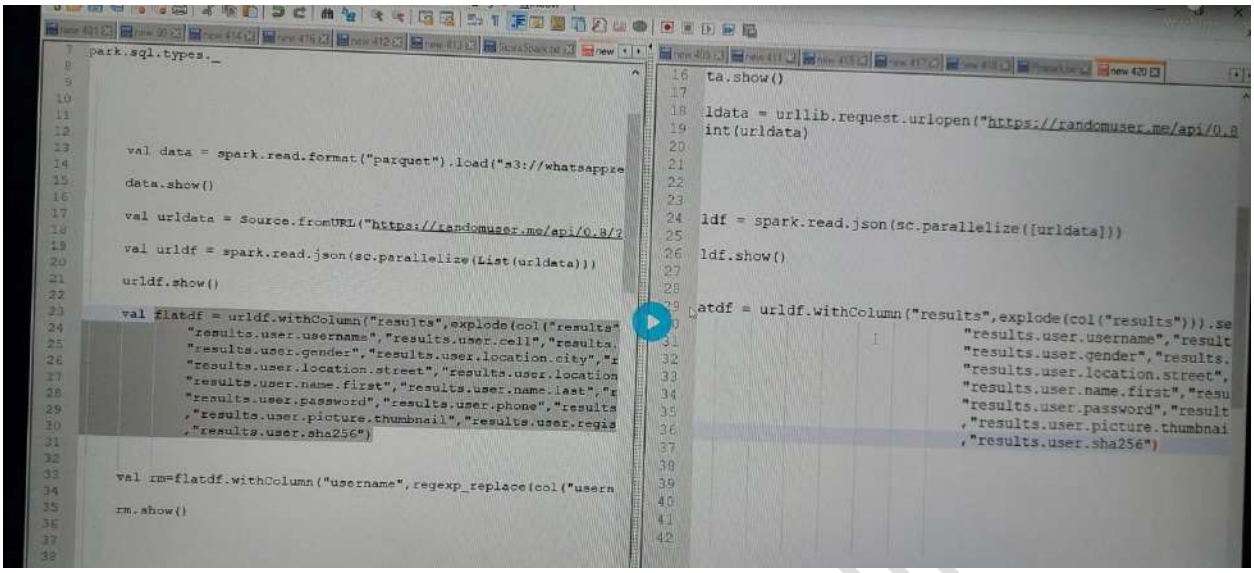
```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import Row
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import .*
import urllib
```

Read the data from Webapi: urlopen



```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import Row
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
import urllib
from urllib import request

data = spark.read.format("parquet").load("s3://whatsappzeyo/src/project.parquet")

data.show()

urldata = urllib.request.urlopen("https://randomuser.me/api/0.8/?results=500").read().decode("utf8
```

Change in parallelize

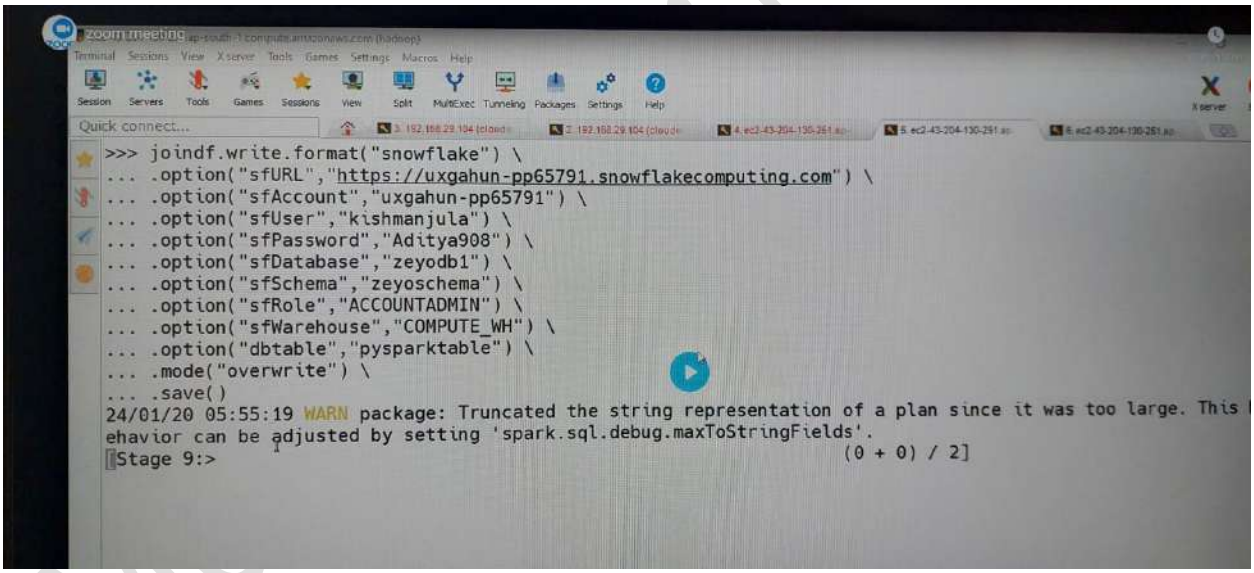


```
park.sql.types._

val data = spark.read.format("parquet").load("s3://whatsappre
data.show()
val urldata = Source.fromURL("https://randomuser.me/api/0.8/?
val urldf = spark.read.json(sc.parallelize(List(urldata)))
urldf.show()
val flatdf = urldf.withColumn("results",explode(col("results"
          "results.user.username","results.user.cell","results.
          "results.user.gender","results.user.location.city","r
          "results.user.location.street","results.user.location
          "results.user.name.first","results.user.name.last","r
          "results.user.password","results.user.phone","results
          ,"results.user.picture.thumbnail","results.user.regis
          ,"results.user.sha256")
val rm=flatdf.withColumn("username",regexp_replace(col("usern
rm.show()
```

```
ta.show()
ldata = urllib.request.urlopen("https://randomuser.me/api/0.8
int(urldata)
ldf = spark.read.json(sc.parallelize([urldata]))
ldf.show()
atdf = urldf.withColumn("results",explode(col("results"))).se
                          "results.user.username","result
                          "results.user.gender","results.
                          "results.user.location.street",
                          "results.user.name.first","resu
                          "results.user.password","result
                          ,"results.user.picture.thumbnai
                          ,"results.user.sha256")
```

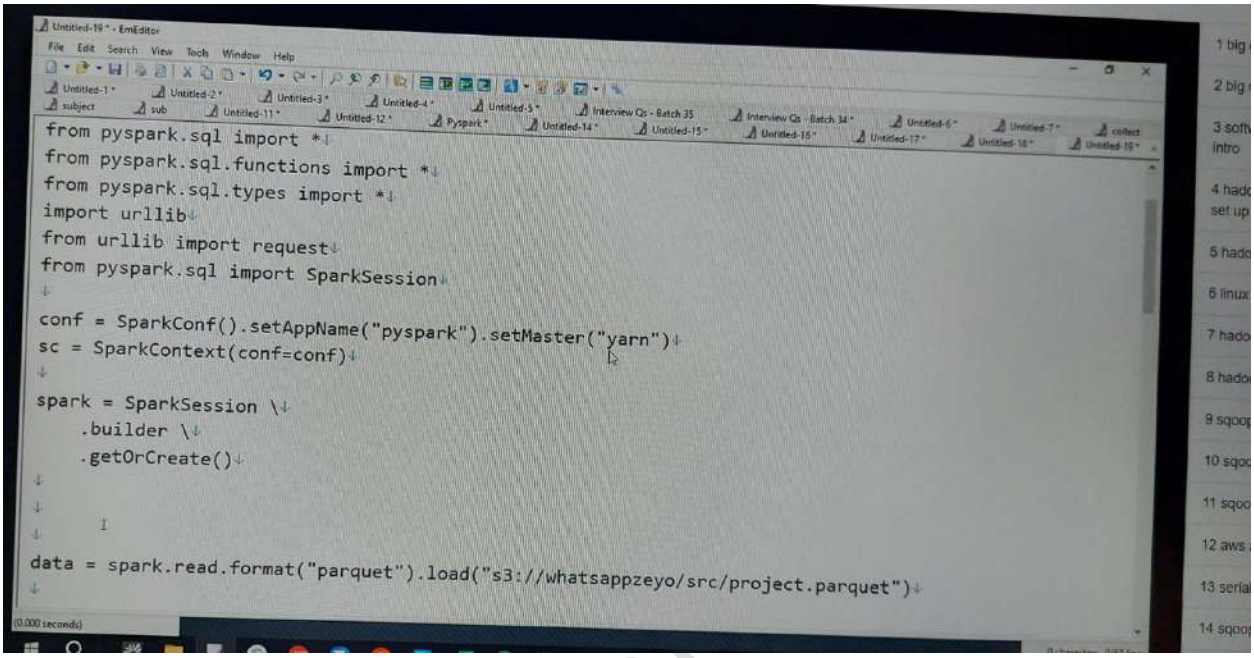
No Seq in python, it should be passed as list

Multiline in python should be ended with \



```
>>> joindf.write.format("snowflake") \
... .option("sfURL","https://uxgahun-pp65791.snowflakecomputing.com") \
... .option("sfAccount","uxgahun-pp65791") \
... .option("sfUser","kishmanjula") \
... .option("sfPassword","Aditya908") \
... .option("sfDatabase","zeyodb1") \
... .option("sfSchema","zeyoschema") \
... .option("sfRole","ACCOUNTADMIN") \
... .option("sfWarehouse","COMPUTE_WH") \
... .option("dbtable","pysparktable") \
... .mode("overwrite") \
... .save()
24/01/20 05:55:19 WARN package: Truncated the string representation of a plan since it was too large. This
ehavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
[Stage 9:>                                                          (0 + 0) / 2]
```

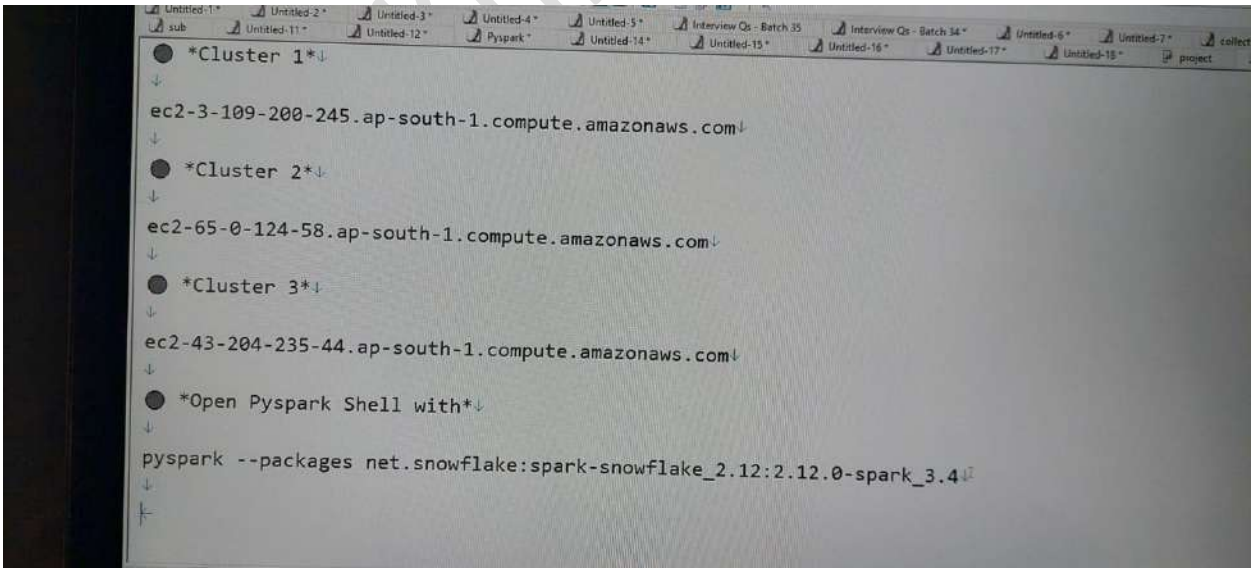
Generating pyfile(similar to jar):
Adding SparkContext and SparkSession in Pyspark code



```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
import urllib
from urllib import request
from pyspark.sql import SparkSession

conf = SparkConf().setAppName("pyspark").setMaster("yarn")
sc = SparkContext(conf=conf)

spark = SparkSession \
    .builder \
    .getOrCreate()


data = spark.read.format("parquet").load("s3://whatsappzeyo/src/project.parquet")
```

Save this file with an extension as .py

Practice on a cluster:



```
*Cluster 1*

ec2-3-109-200-245.ap-south-1.compute.amazonaws.com

*Cluster 2*

ec2-65-0-124-58.ap-south-1.compute.amazonaws.com

*Cluster 3*

ec2-43-204-235-44.ap-south-1.compute.amazonaws.com

*Open Pyspark Shell with*

pyspark --packages net.snowflake:spark-snowflake_2.12:2.12.0-spark_3.4
```

exit() - to exit the shell

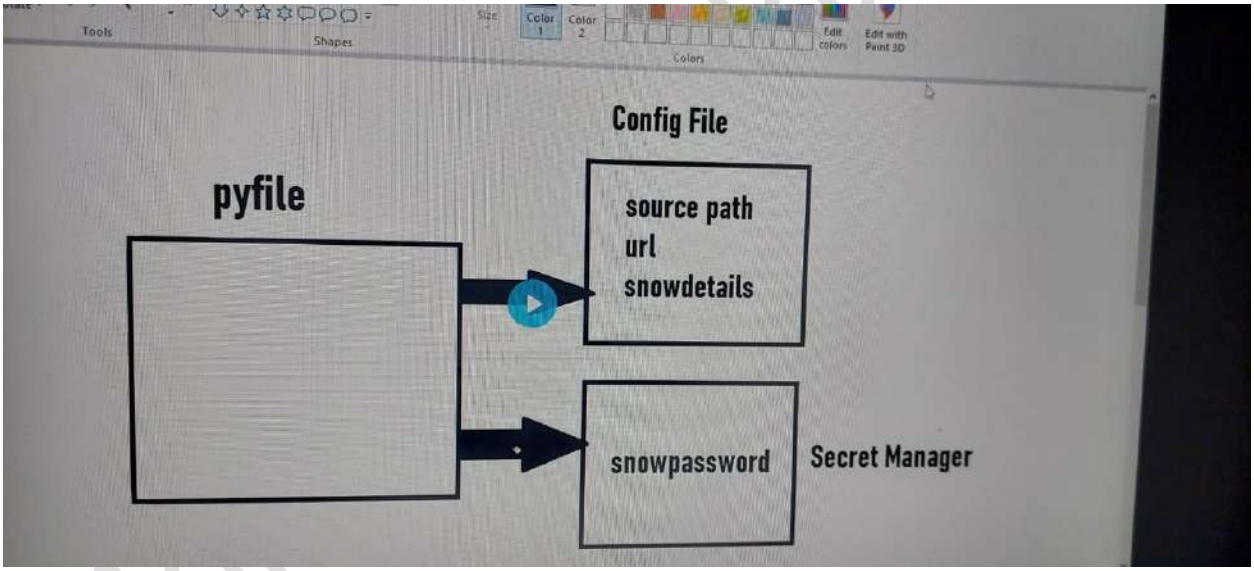
Pyspark EMR deployment Secret manager Config File
Agenda Day 43

AWS is an on-demand cluster, we can't stop a cluster, we can just start a cluster when required and terminate, where as GCP dataproc supports to stop a cluster.

Pyspark execution:

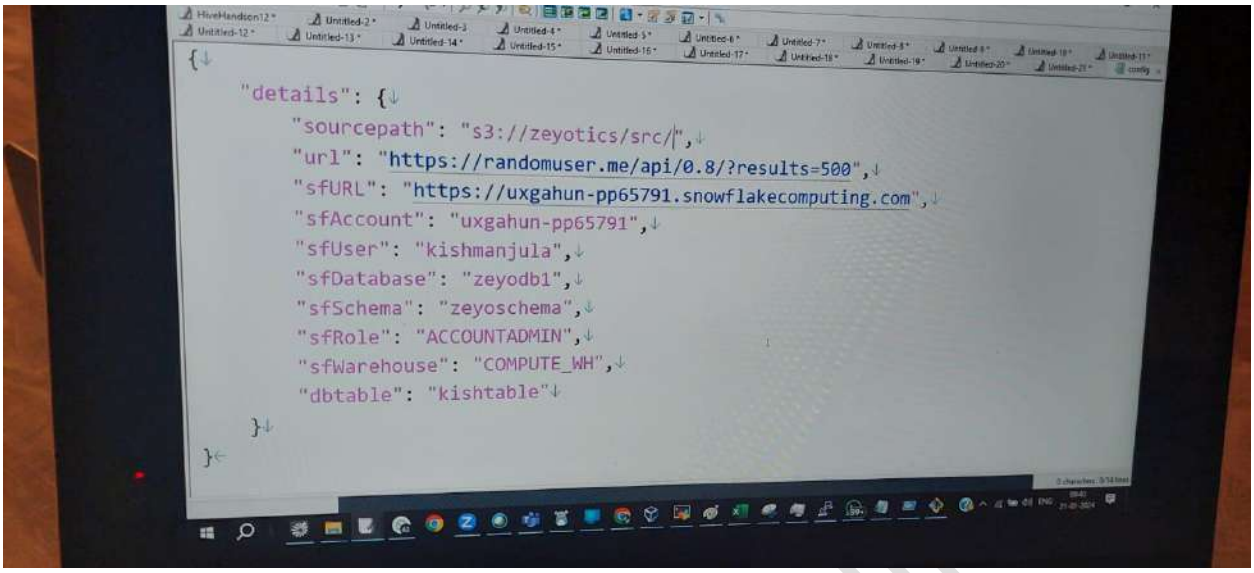
the code should be saved as a .py file in S3 and give that in the step function of EMR.

All the config details in the python code should go to config file and the passwords to be stored in the Secret Manager.



Steps to add configuration to the cluster:
1) create a cluster
2) create a config file as config.json

3)Sample Config file:

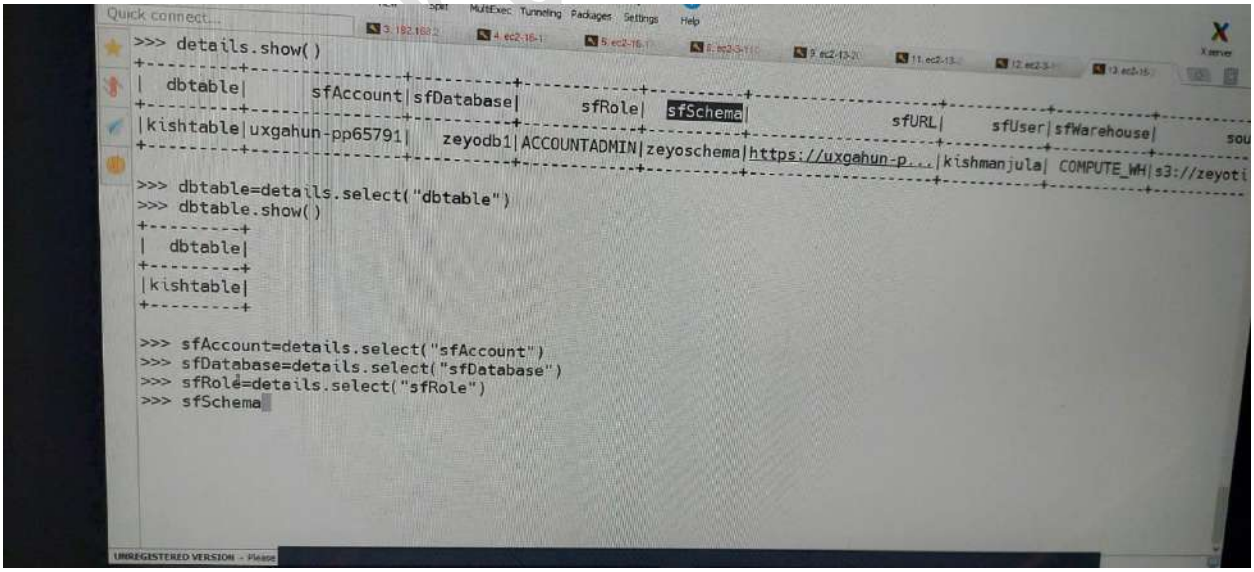


4) Load the config to S3

5)Connect to the cluster and start pyspark

6) Reading the config file.

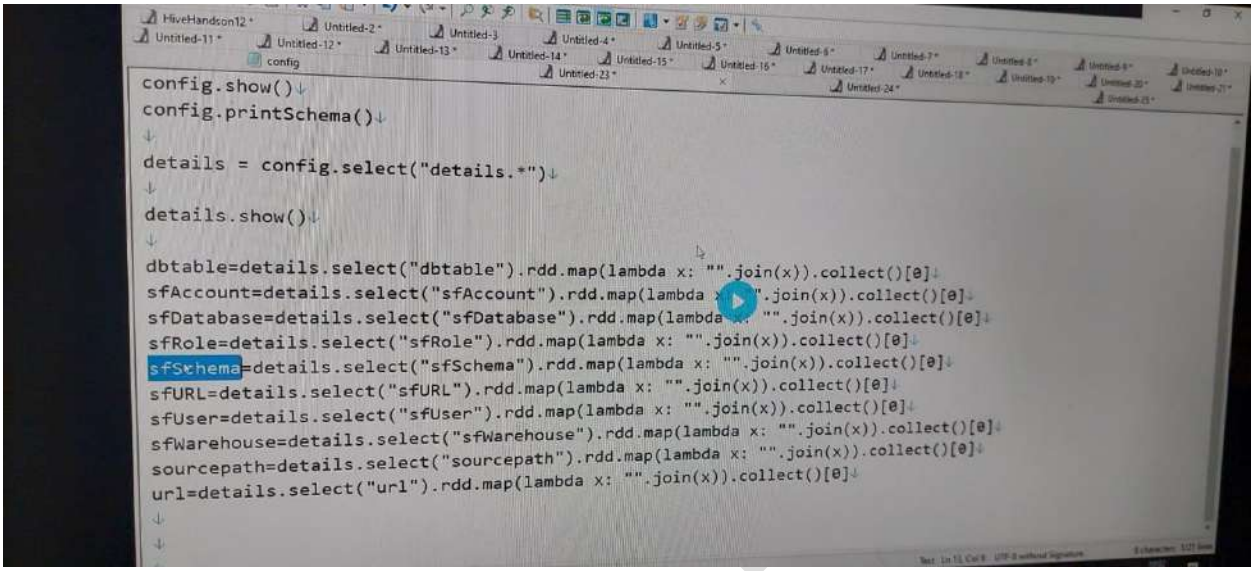
```
config = spark.read.format("json").option("multiline","true").load("s3://zeyotics/config.json")
details = config.select("details.*")
details.show()
dbtable = details.select("dbtable")
```



dbtable is as a dataframe, but we need a string. Converting dataframe to string.

dbtable = details.select("dbtable").rdd.map(lambda x: "".join(x)).collect()[0]

print(dbtable)

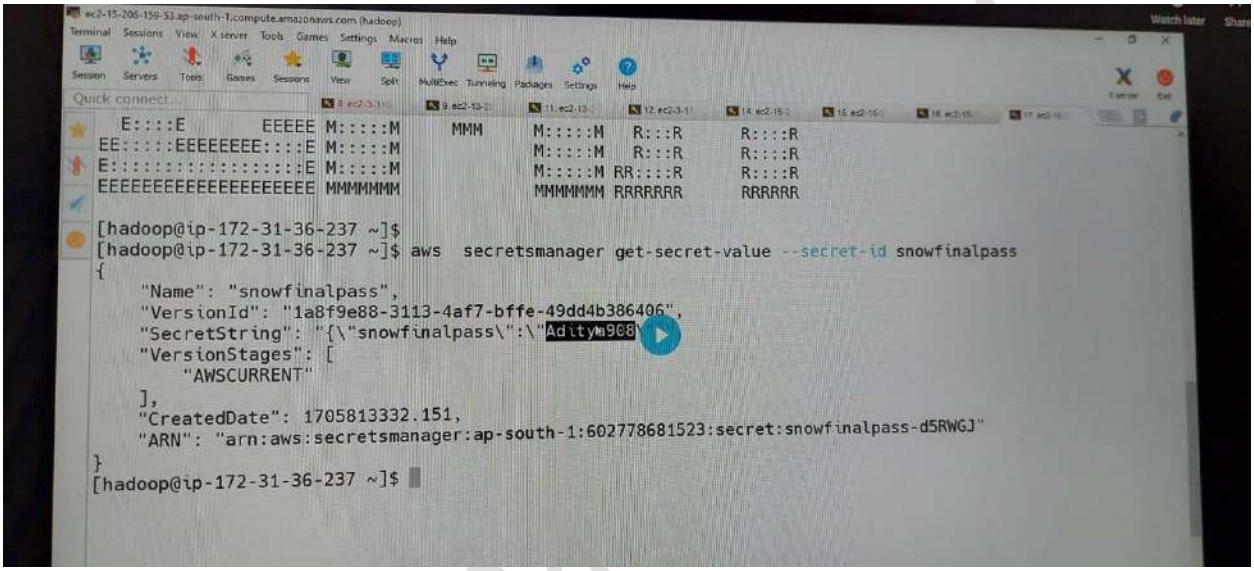


now replace with these variables in the source code in the relevant places

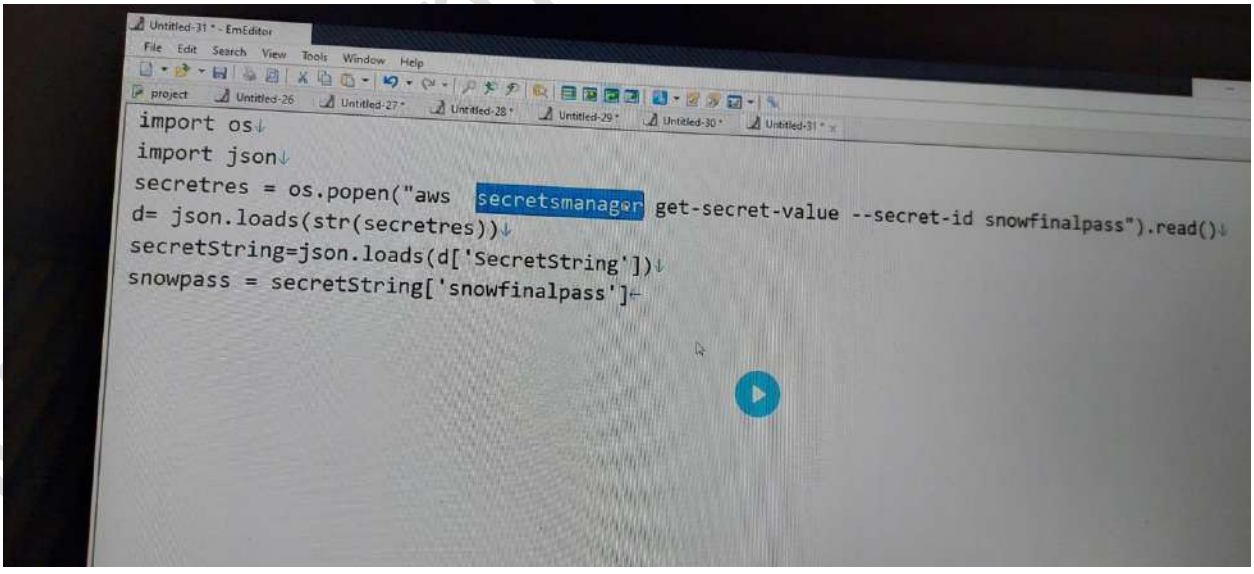
Passwords will be stored in Secert Manager(AWS Service) and will retrieved and used in the pyspark code.

Steps to use this password in Secret Manager in the python code

1) Add role permissions to the secret managers' ARN in IAM
2) Command to get the password from the secret manager

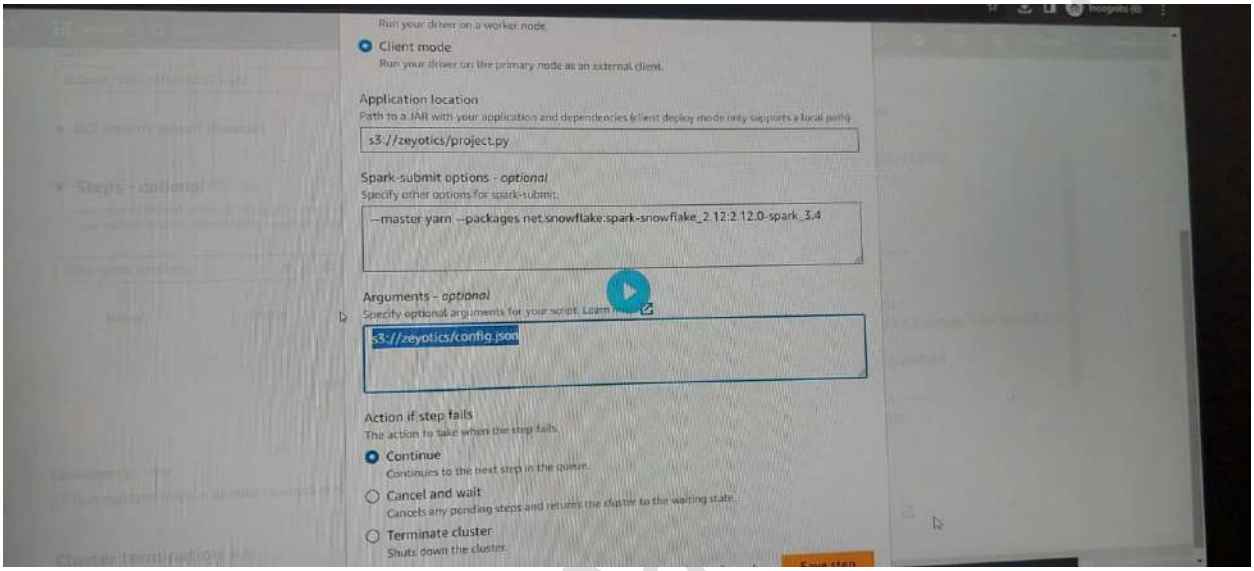


3) Python code to get the password



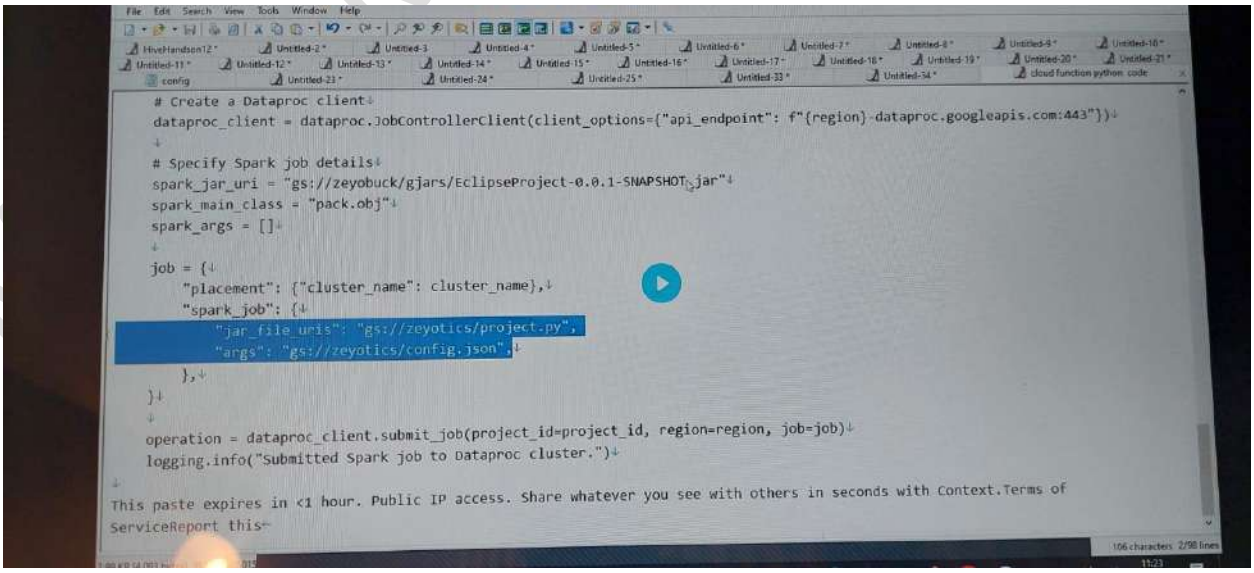
We can use the variable snowpass in our python code.
Getting the config path the args.
import sys
configpath = sys.argsv[1]

And the arguments should be given while creating the cluster.



Signs of pyspark development in GCP

1) Secret Manager API for secret Manager
2) Cloud Storage for storing.
3) Cloud function to Automate – python code



Signs of pyspark development in Azure

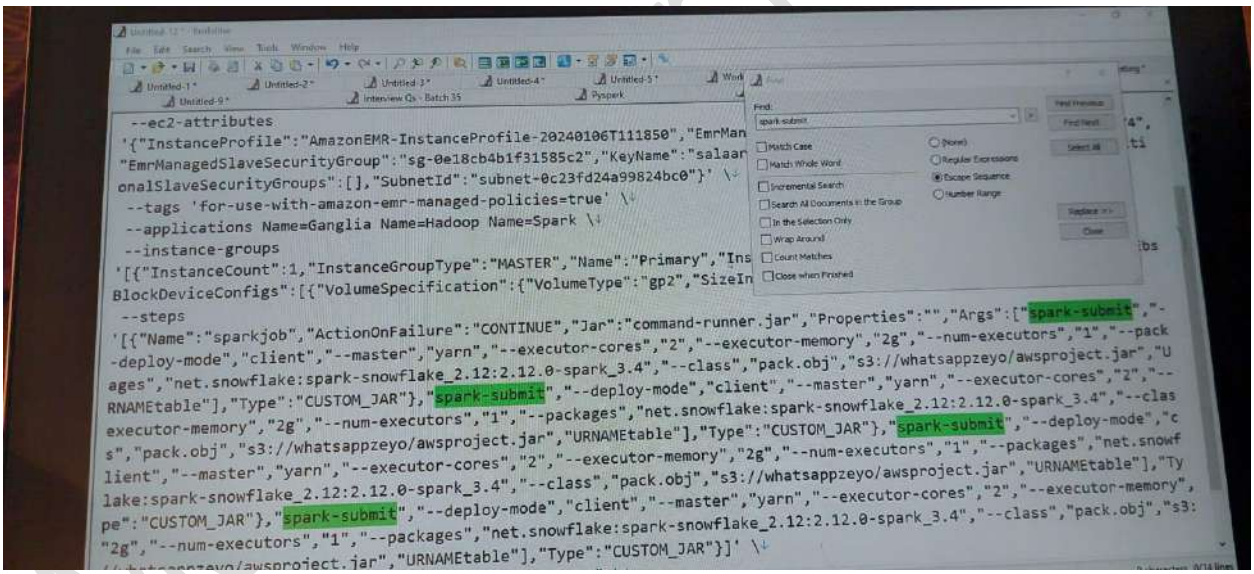
1) Blob – Storage
2) Data factory – executes the same.

Nifi Intro, Data Flow, Streaming Spark
Agenda Day 44:

Nifi Download link: https://archive.apache.org/dist/nifi/1.6.0/nifi-1.6.0-bin.zip

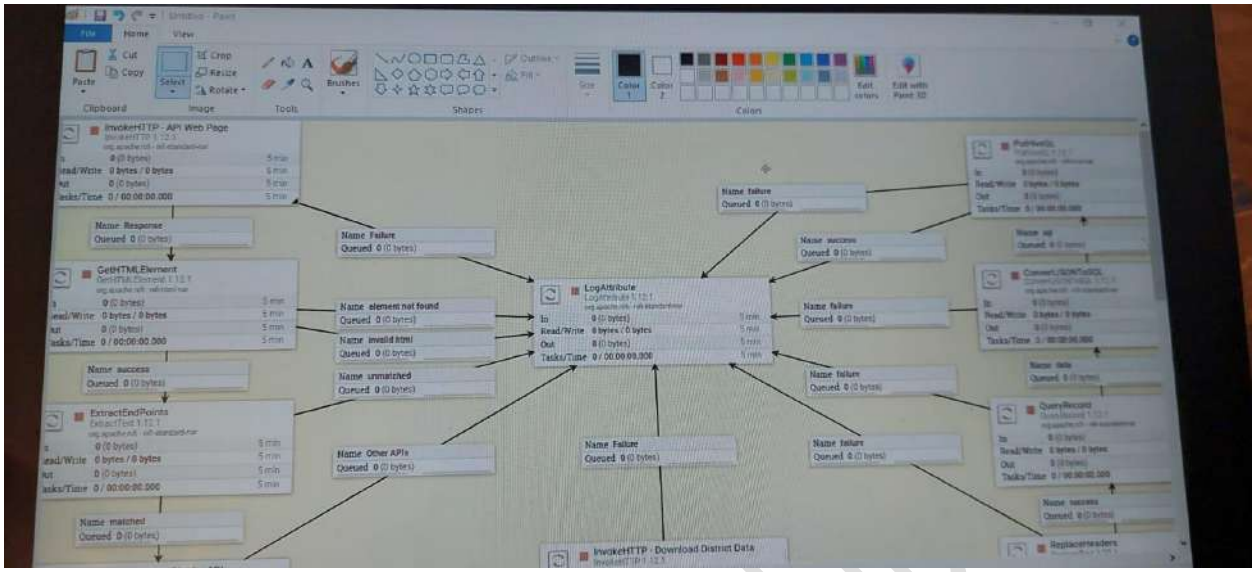
Orchestrate the jobs → this means executing the jobs one after the other.
Tools: Airflow, Talend, Nifi, Oozie, ADF, AWS DP

in the AWS CLI Command, we can give multiple spark-submit jobs:



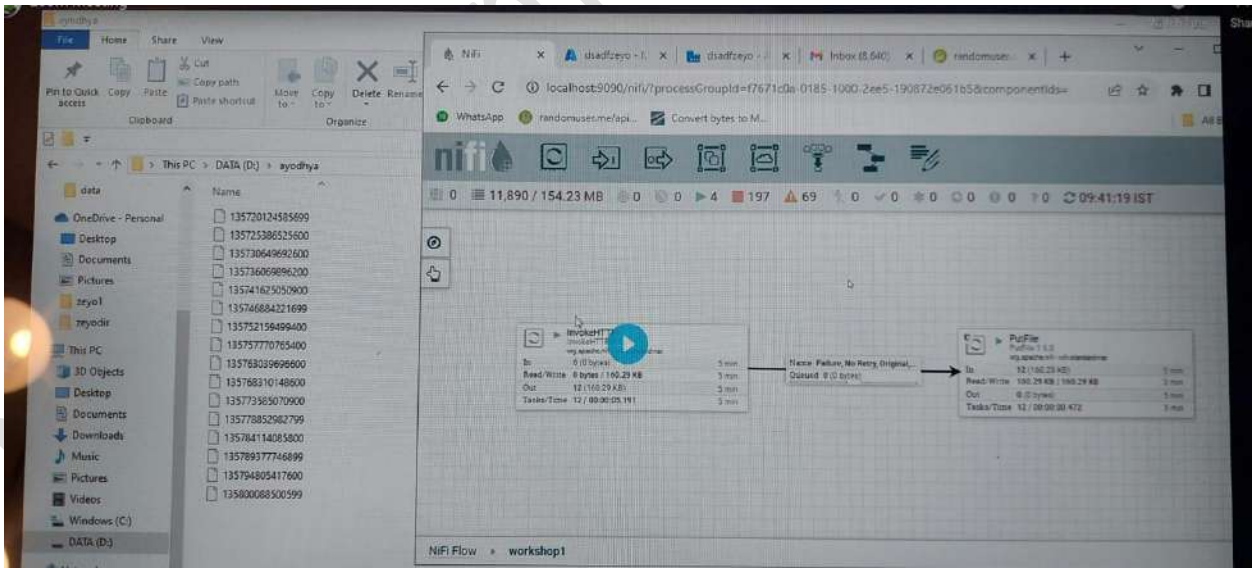
Features of Orchestration (Step by Step Process)Tools:
1) Arranging the Jobs
2)Scheduling the Jobs
3)Data Flow

Below is an example of orchestration using nifi (Step by step arrangement)



distcp – Used to copy data from one Hadoop cluster to another Hadoop cluster

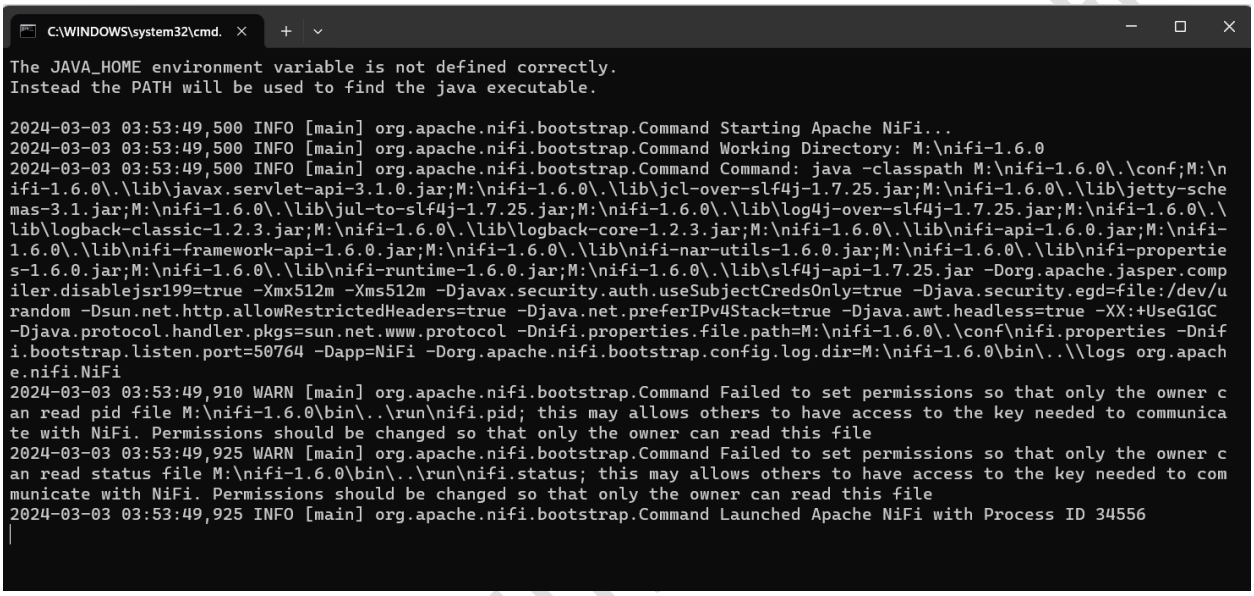
Simple Orchestration Example:
Arrangement: InvokeHTTP and put file are configured and connected
Scheduling Job: Job is scheduled to be executed for every 5sec
Dat flow: data is flowing to Ayodhya Folder



Nifi:

1) Windows – unzip and place it directly under any main drive
2)unzip <zip_file_path>

to launch nifi, go to nifi folder → bin and run run-nifi batch file
for nifi, we need Java 8 (1.8)

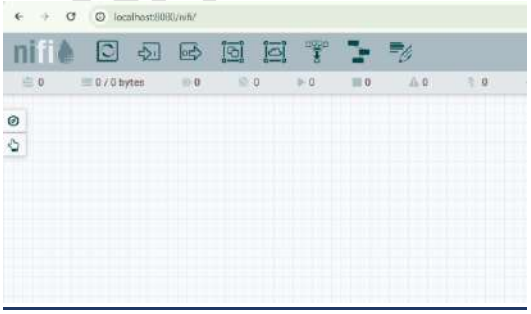


To modify the existing java, if you have other than 1.8,

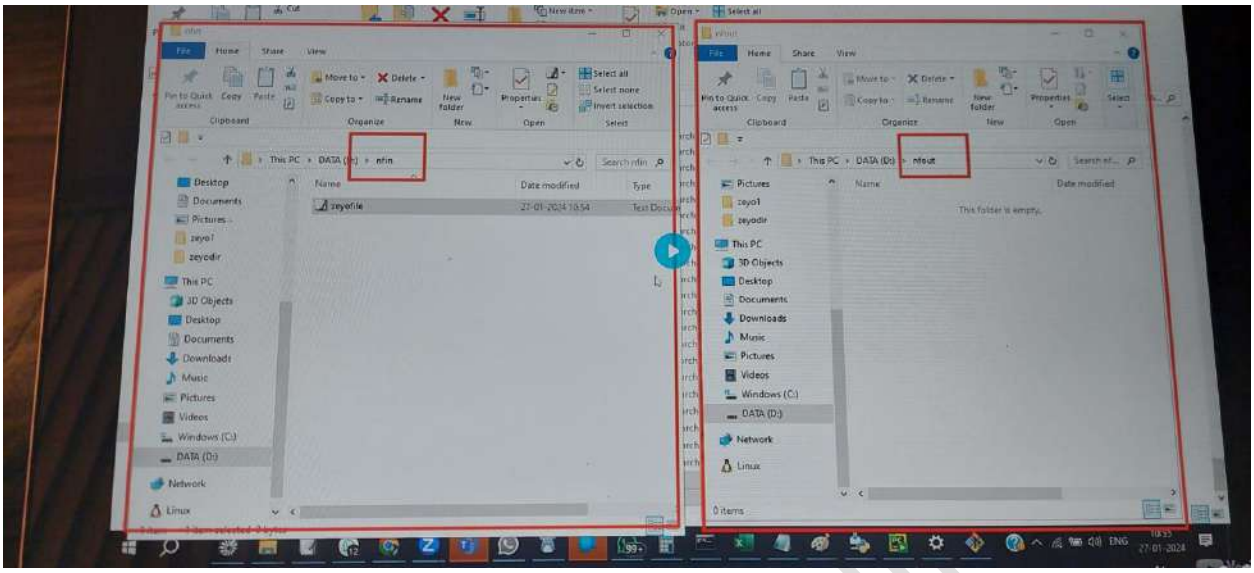
Go to Winodws → Add or remove programs → remove any other java versions downloaded.

Command to start nifi: **sh nifi.sh start** from nifi bin folder

Nifi Handson:
localhost:8080/nifi



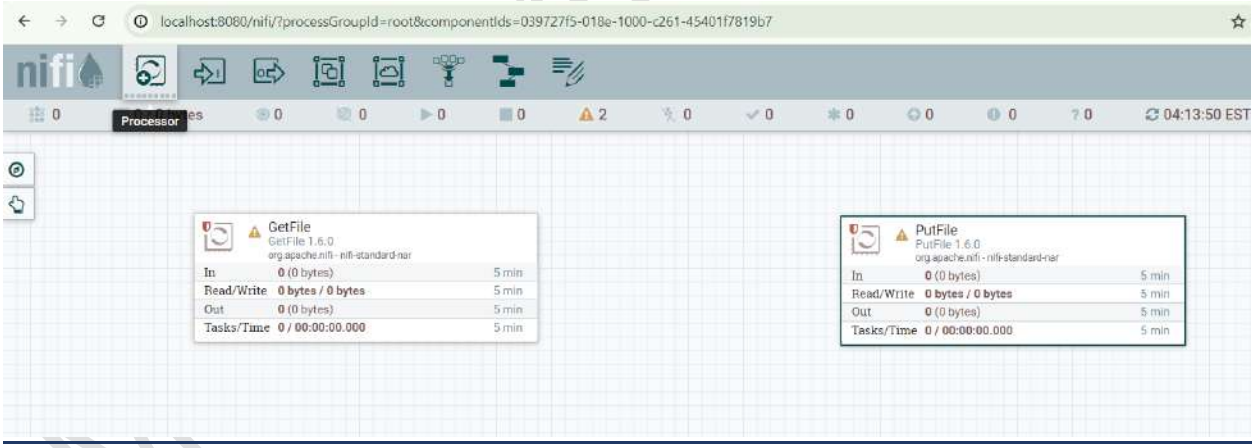
Sample Execution with Nifi



Files in the nfin should flow to nfout

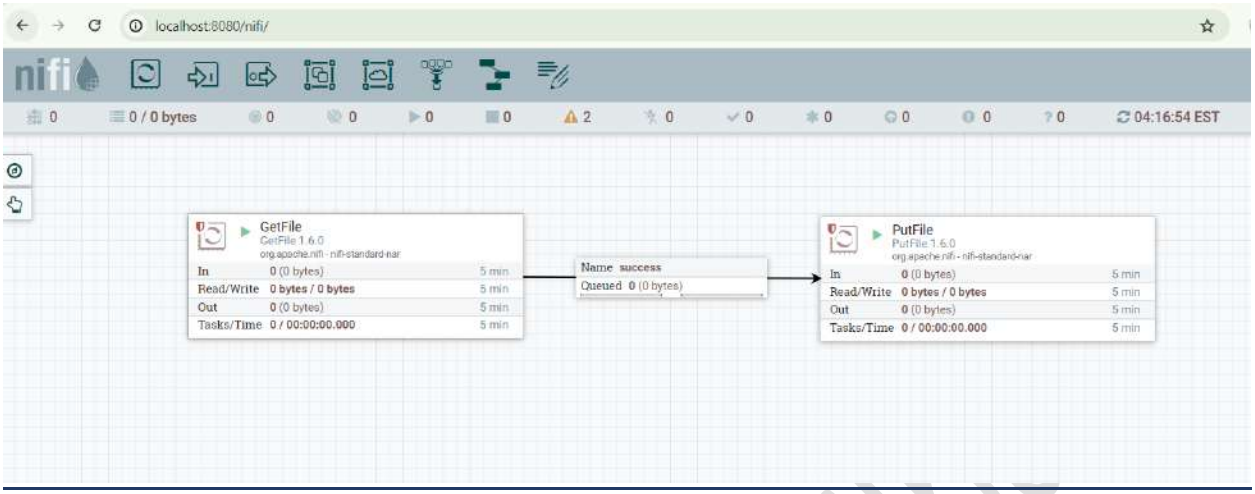
Steps to do it:

1) Add two processor in nifi (getFile) and also (putfile) and connect them

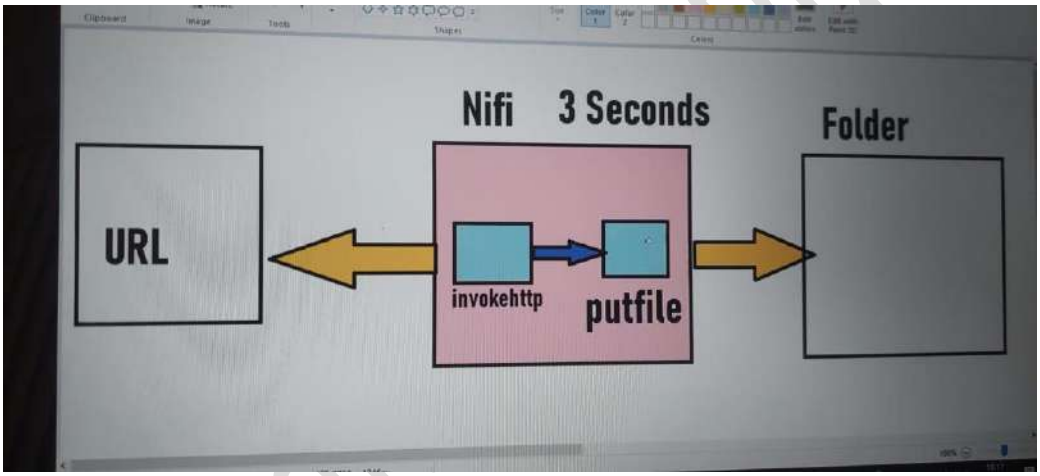


2)Double click getfile processor -> go to properties and give theinput directory and also for putfile give the output directory.

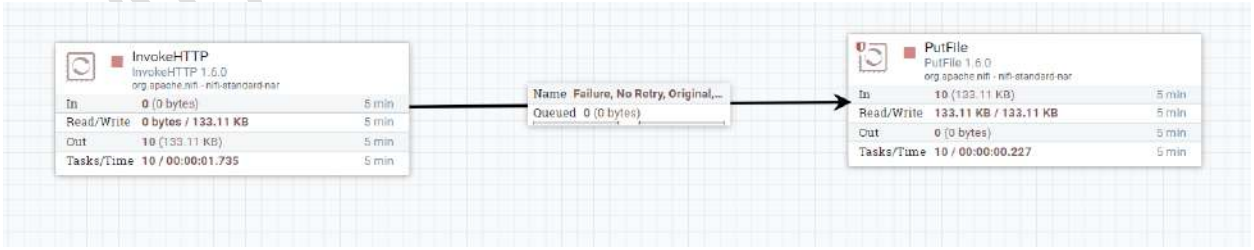
3) if the processor is the last in the flow, click failure and success checkboxes on the right side in the settings tab.



Scenario with Nifi:

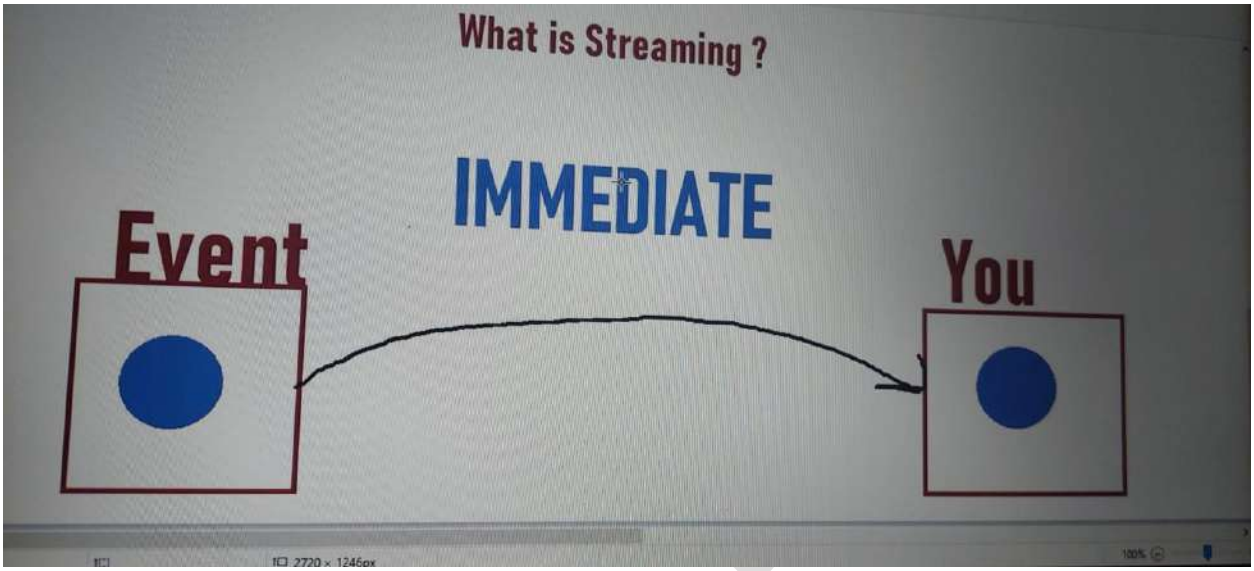


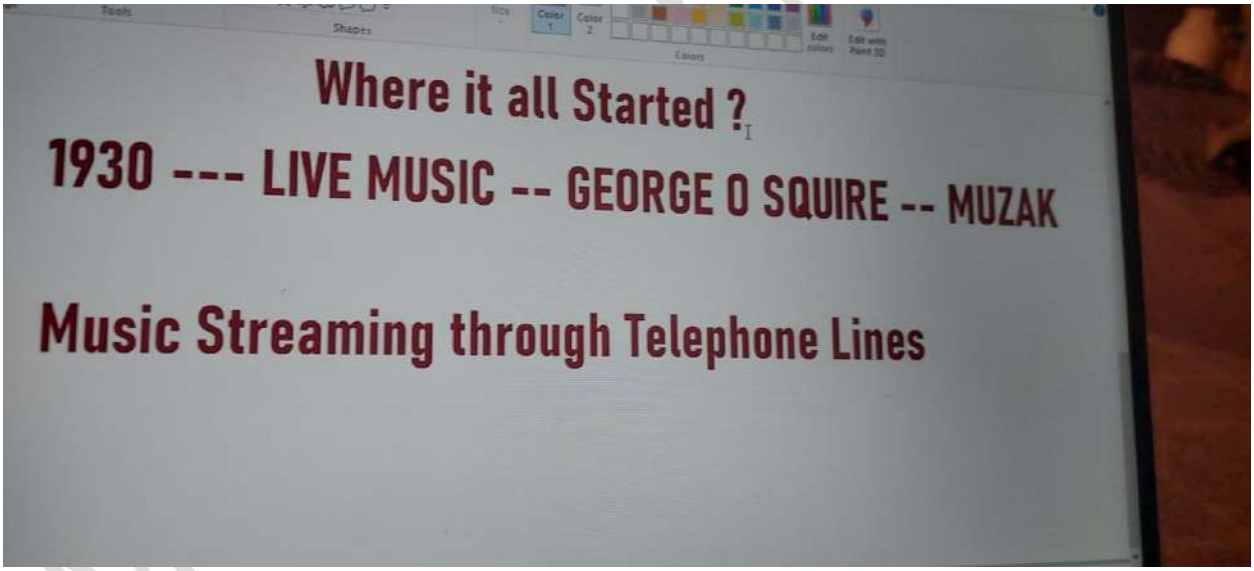
url: https://randomuser.me/api/0.8/?results=10



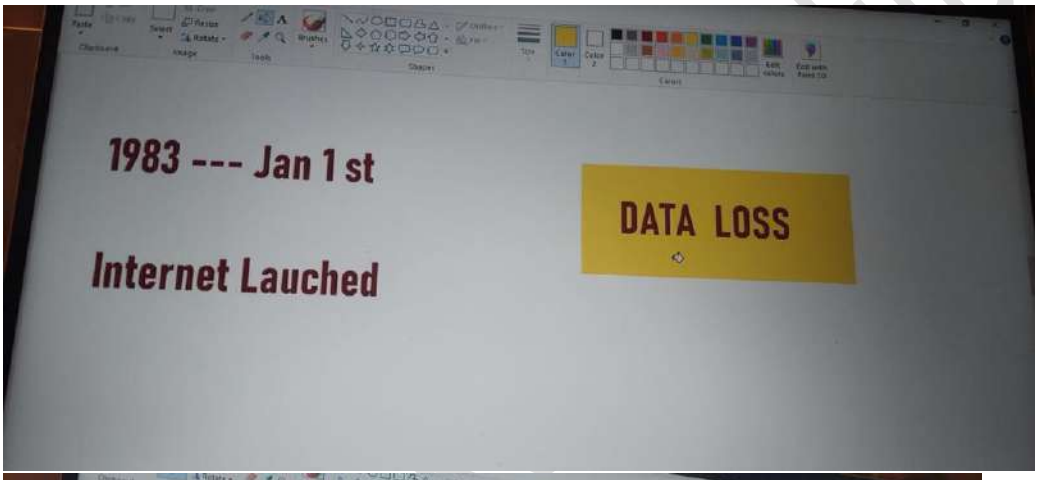
invokeHTTP: Cosumes data from a URL. URL can be given in properties (Remote URL) and scheduling can be configured in Scheduling tab.

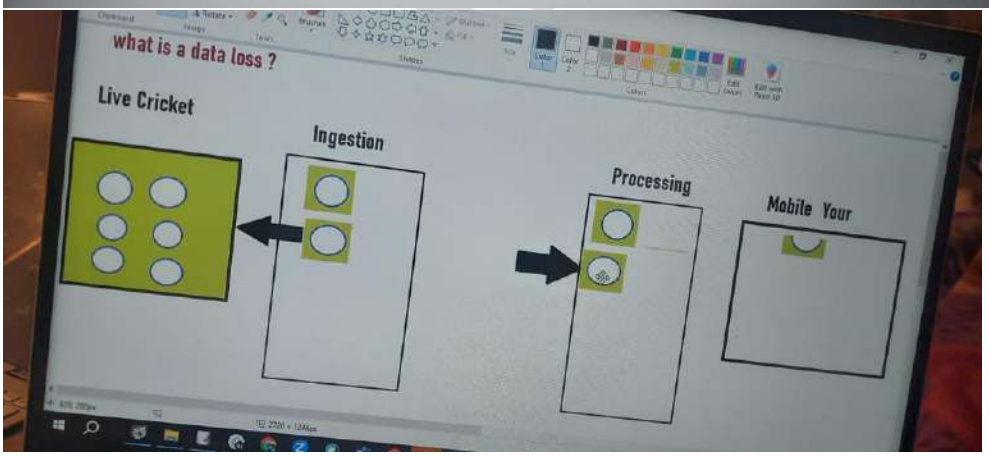
Streaming Intro Workshop:
Agenda Day 45

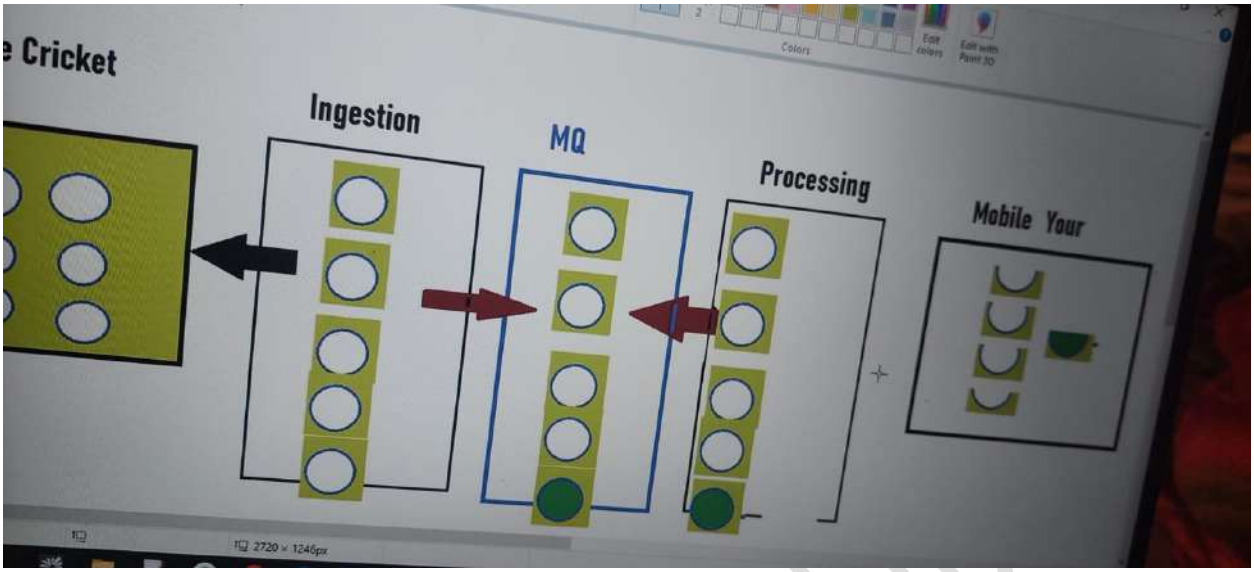


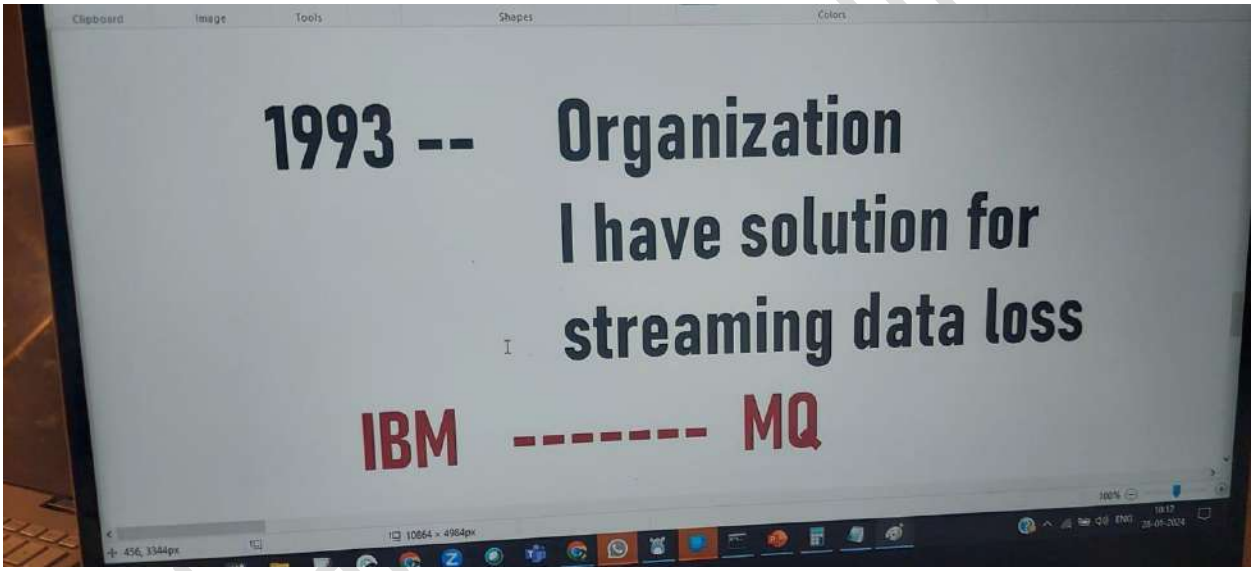


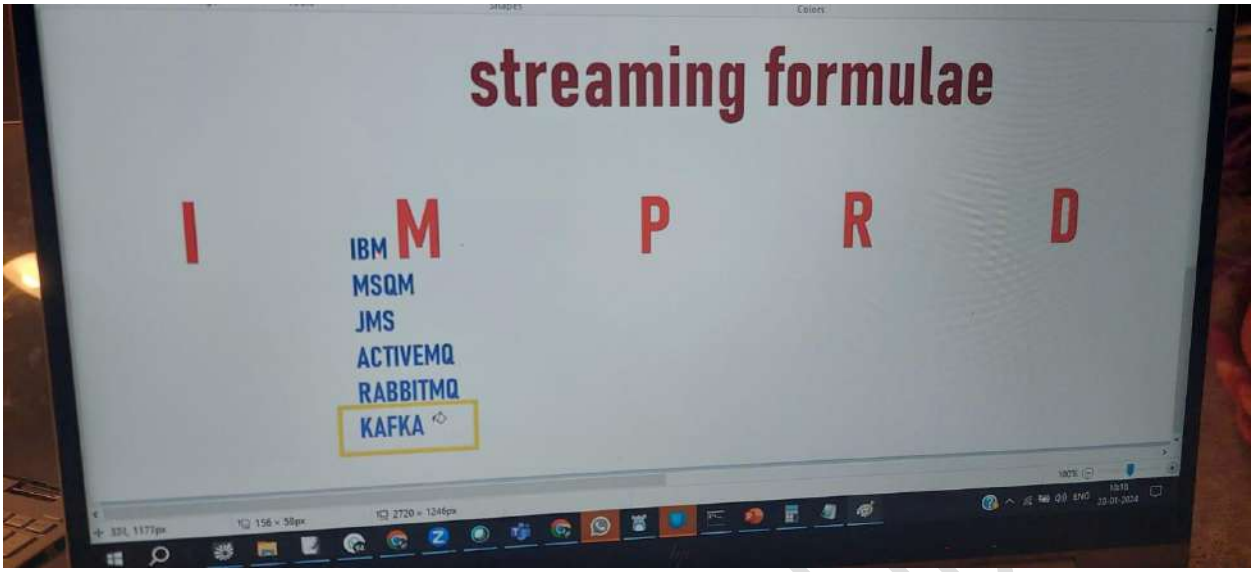
1975 – Microsoft






Cricket
Ingestion
MQ
Processing
Mobile Your


1993 -- Organization
I have solution for
streaming data loss
IBM ------- MQ



Injestion – Message Queue – Processing – Real Time – database

Kafka Streaming Nifi Topic Data Streaming
Agenda Day 46

Streaming Intro Deep

Amazon kinesis – AWS tool used for Message Queuing

1930 – Live Music Through Telephone lines
1948 – AT & T – Failed phone lines online meeting
1975 – Microsoft
1983 – Internet
1993 – Data loss is the biggest threat

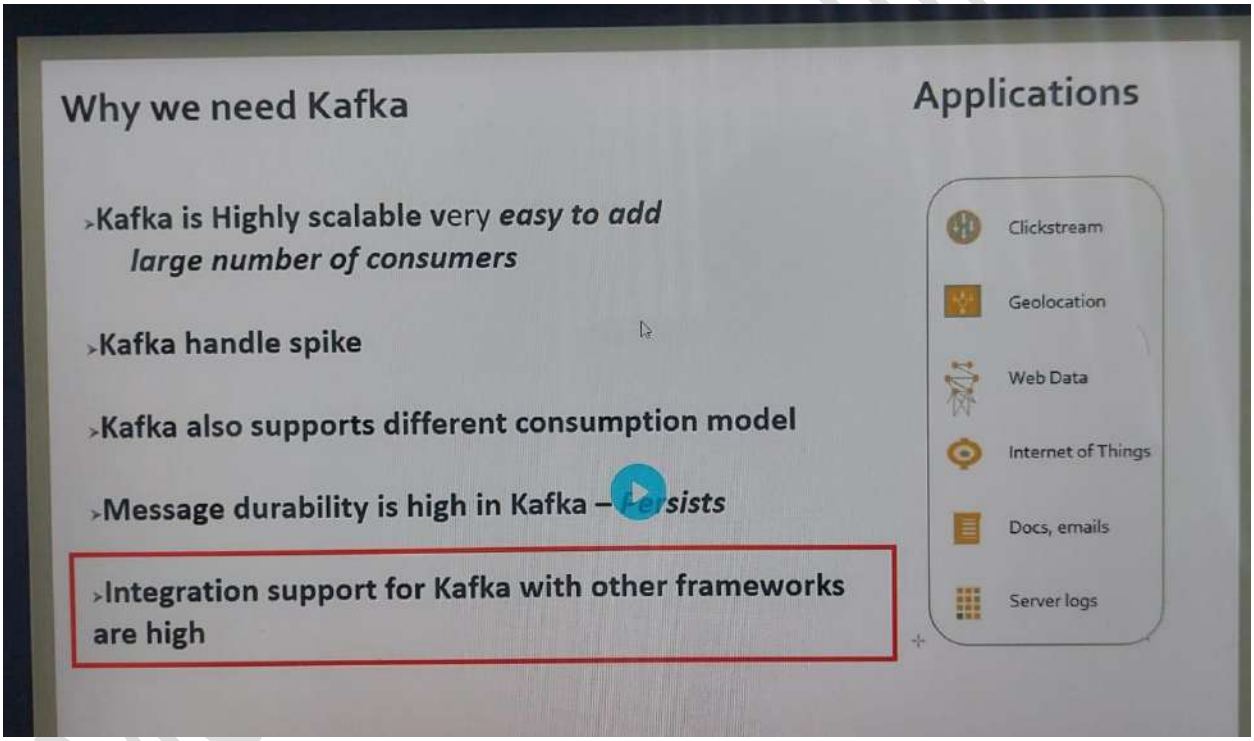
Different MQ came into market
1998 –micorsoft – msmq
1999 – JMS
2004 – ACTIVE MQ
2010 – RABBIT MQ
2009 – FLUME

2011 – Kafka (king of MQ Services)

Message Queue Features:
1) Immediate Availability
2)Incremental is taking Care

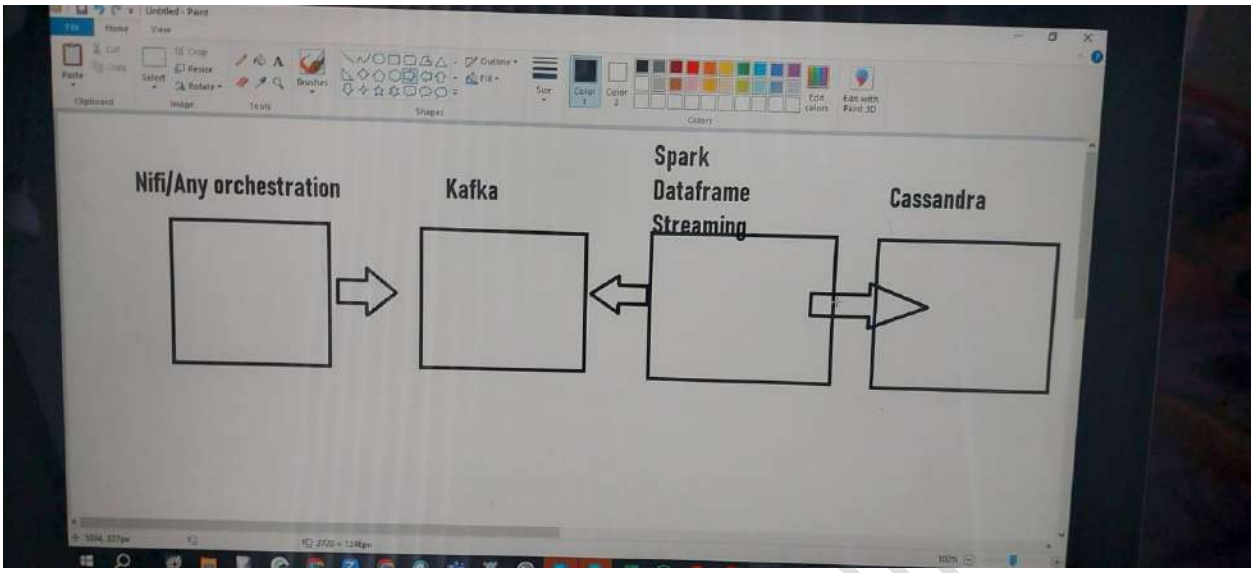
Kafka Intro

1)Open Source – Free of cost
2)Distributed -- Cluster
3)Publlish Subscriber (Producer and Consumer)
4)Message Queue

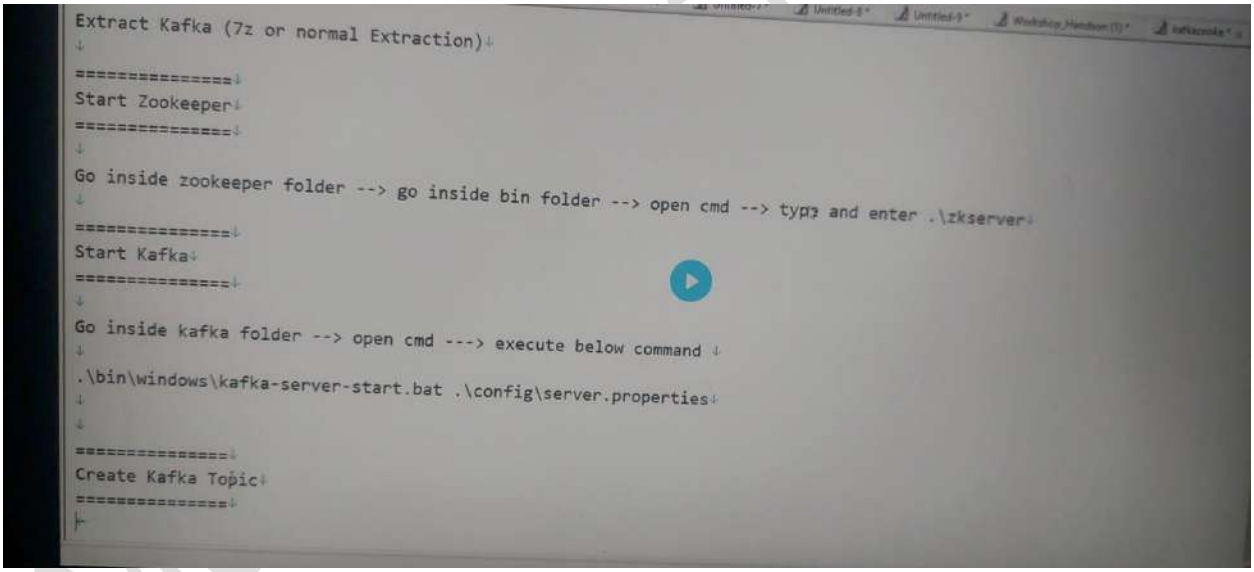


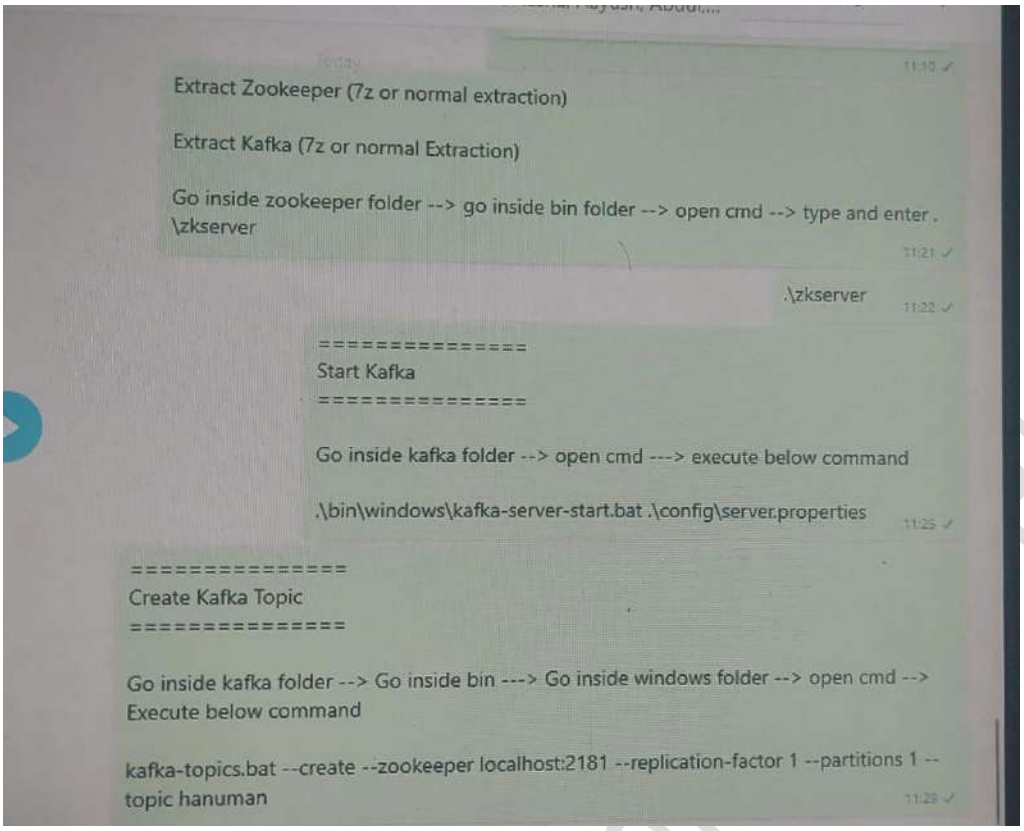
Default retention period for Kafka MQ ia 172hrs, we can customize this in server.properties file in Kafka config folder

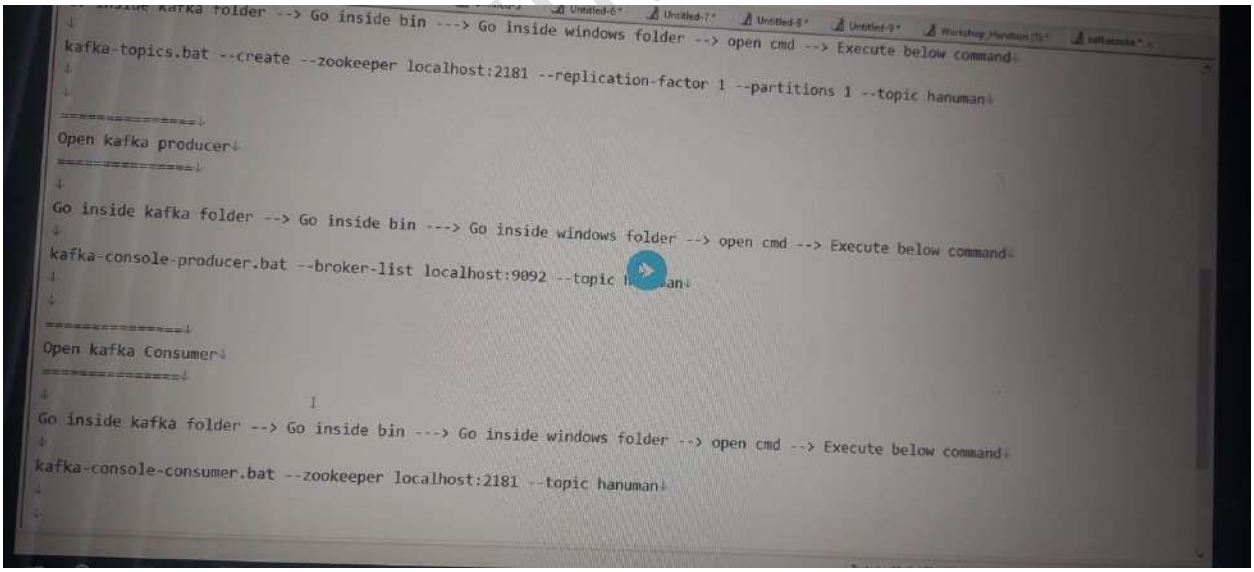
Best Streaming Stack:



Kafka Hands-on







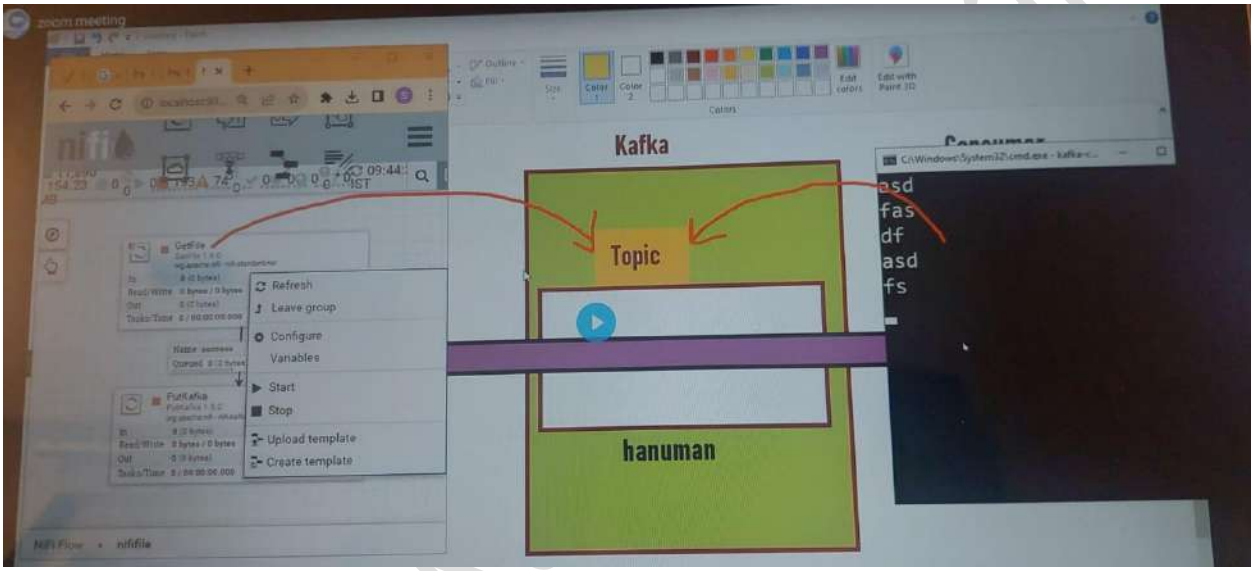
Close all the CMD windows and remove tmp folder created in the drive where kafka is created.

Kafka Spark Integration Streaming
Agenda Day 47

Kafka version – kafka_2.11-0.11.0.0
Zoo keeper version – 3.4.12

Kafka Nifi Integration

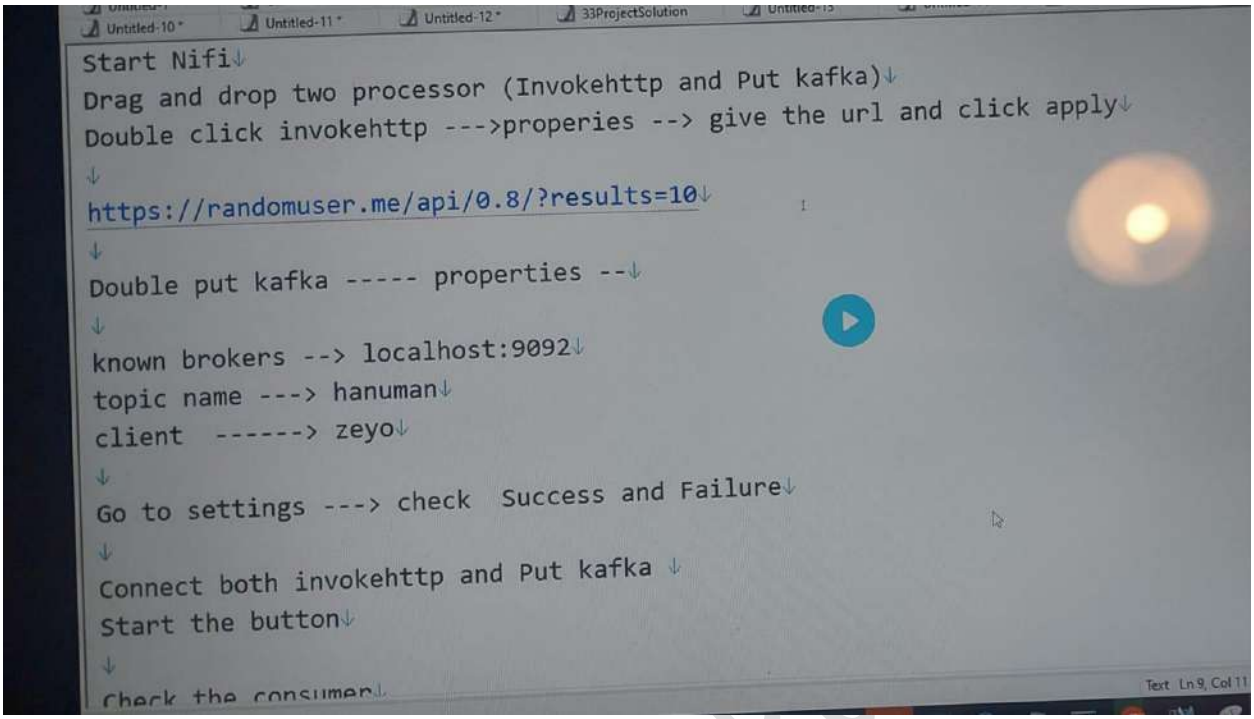
Nifi Should get the data from a file and put that to kafka.



Expected output:
Any file sent to the directory from which the Nigi getFile is consuming should be displayed in the kafka consumer CLI and the file should get automatically deleted from the folder.

Scenario 2:

Nifi should consume from the webURL and produce that to kafka,

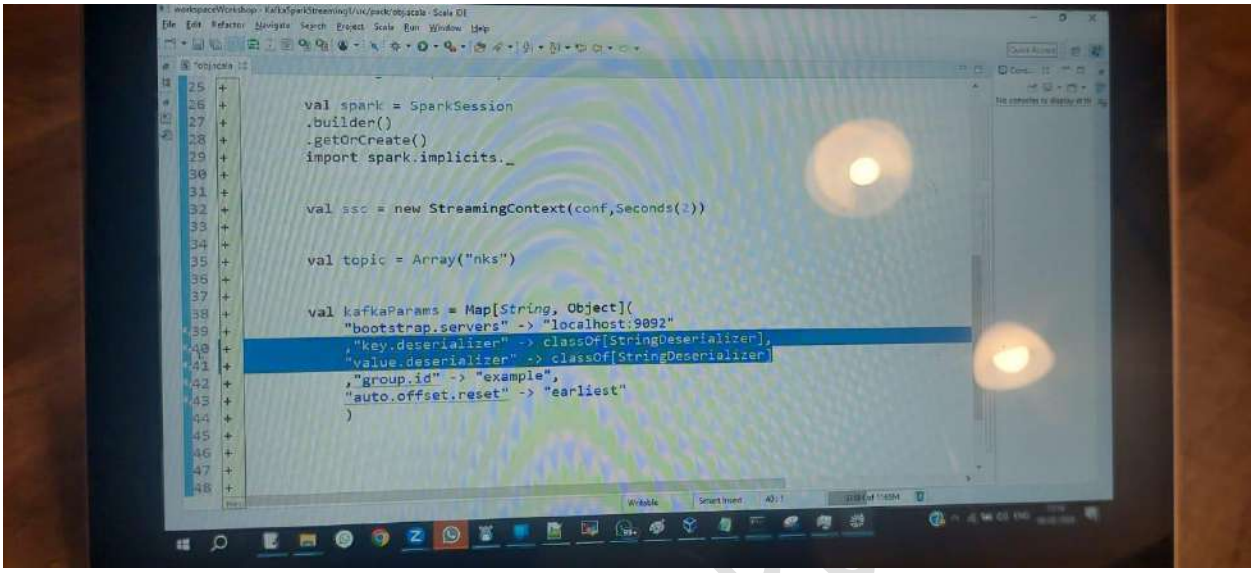


Kafka Tools Setup

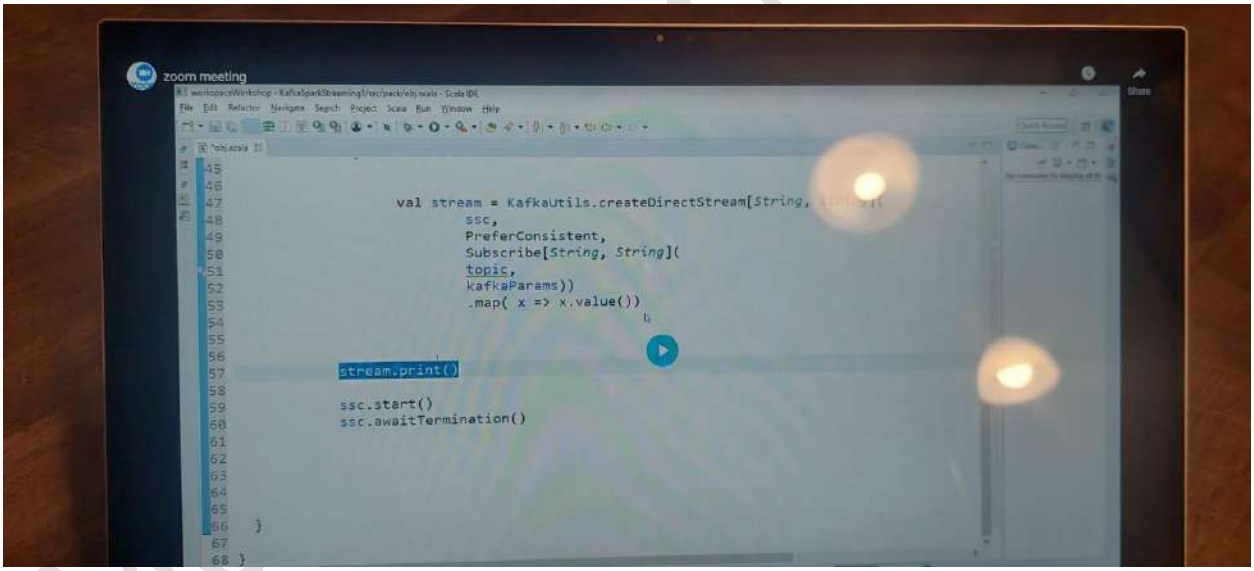
Offset Explorer – helps us to see the kafka messages

Kafka Spark Integration

RDD Streaming and Dataframe Streaming are two ways streaming can be processed.



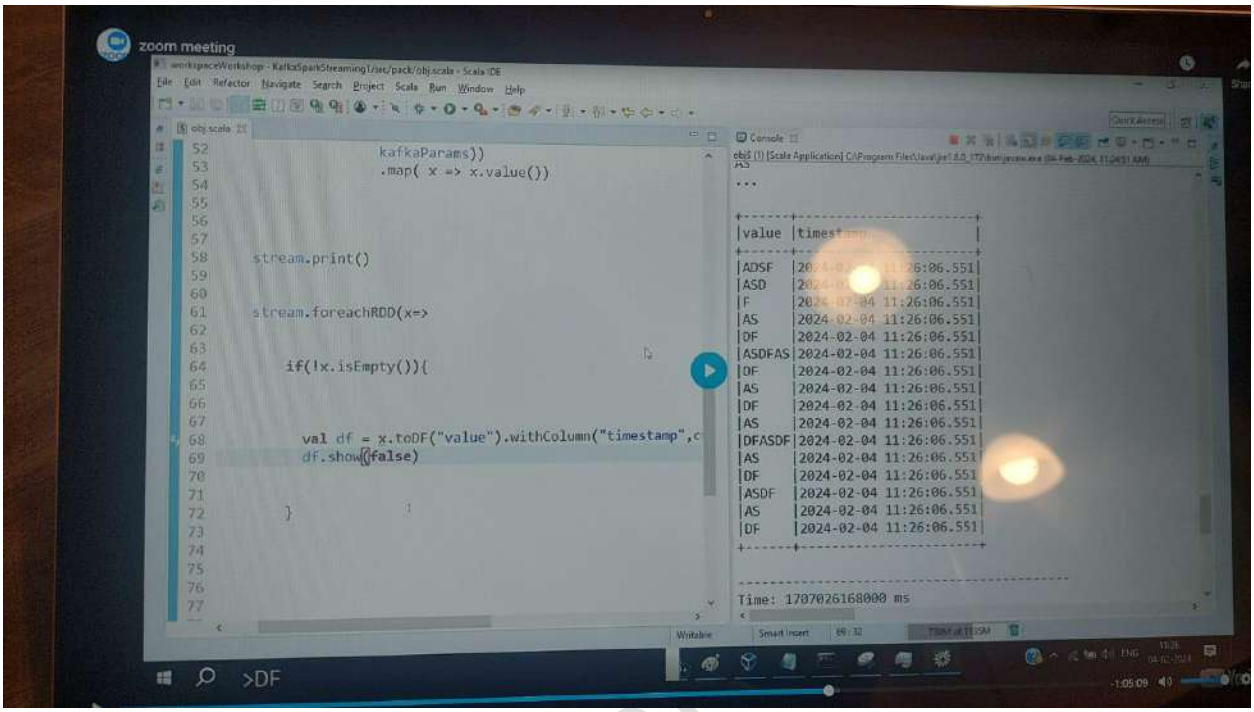
```
val spark = SparkSession
.builder()
.getOrCreate()
import spark.implicits._

val ssc = new StreamingContext(conf,Seconds(2))

val topic = Array("nks")

val kafkaParams = Map[String, Object](
    "bootstrap.servers" -> "localhost:9092"
    ,"key.deserializer" -> classOf[StringDeserializer],
    "value.deserializer" -> classOf[StringDeserializer]
    ,"group.id" -> "example",
    "auto.offset.reset" -> "earliest"
    )
```



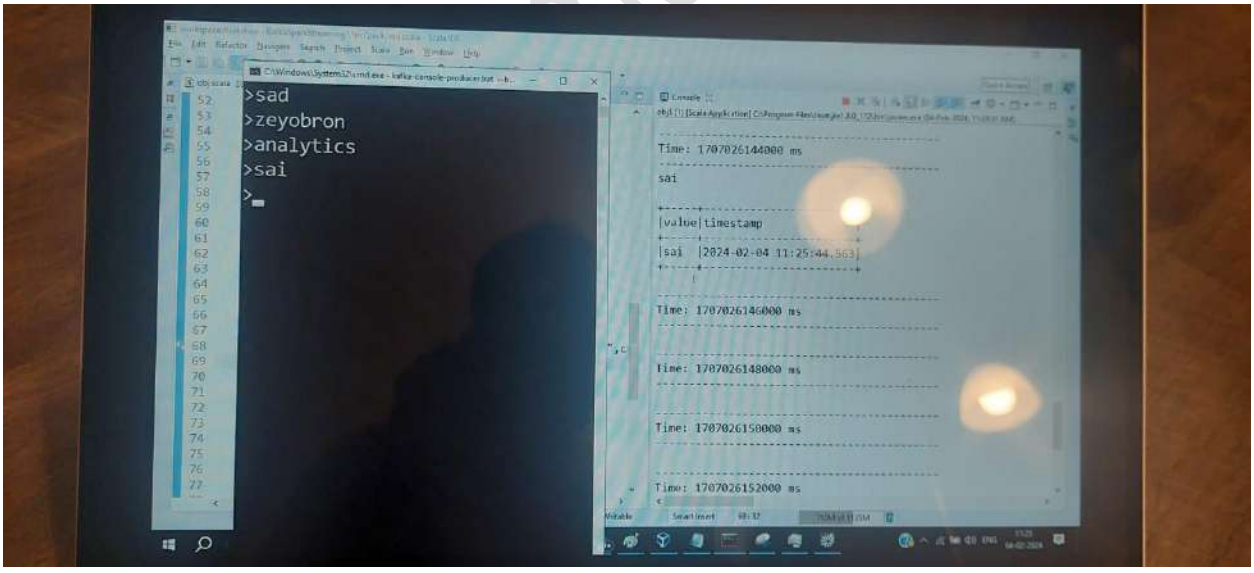

```
val stream = KafkaUtils.createDirectStream[String,
        ssc,
        PreferConsistent,
        Subscribe[String, String](
        topic,
        kafkaParams))
        .map( x => x.value())

stream.print()

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

Converting the stream data produced to kafka to dataFrame spark

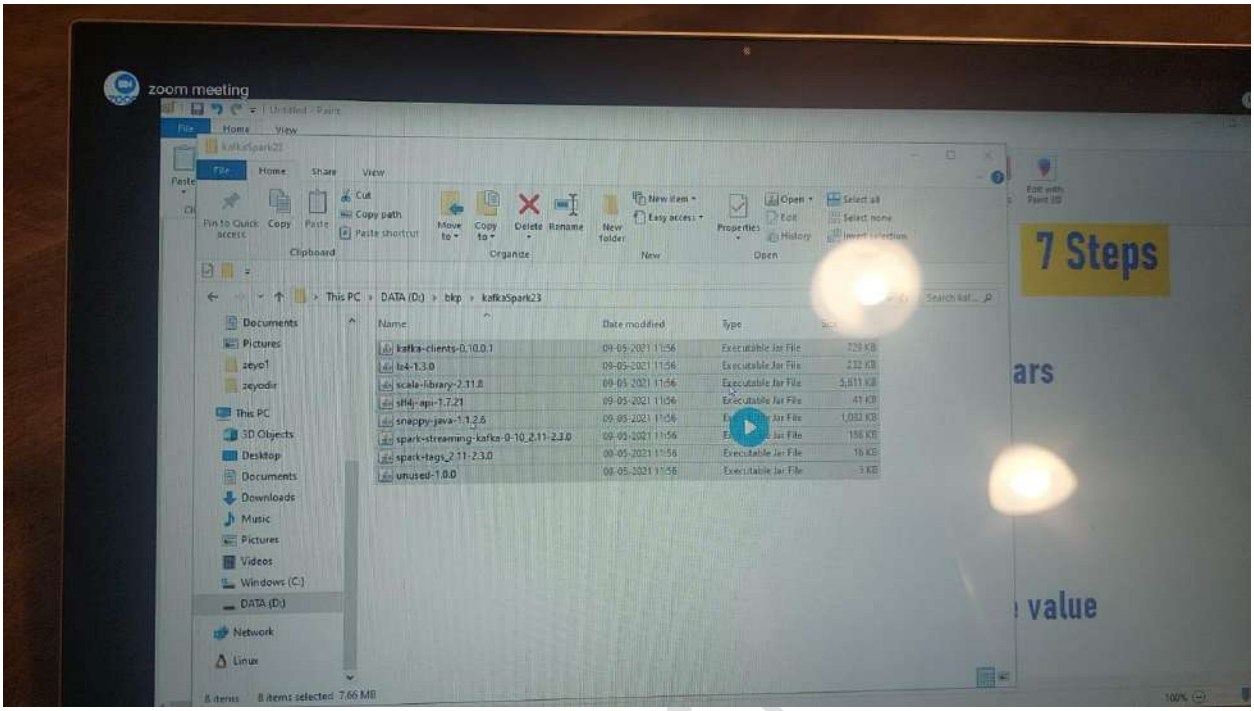
Add Kafka Related Jars:



Output:



Kafka Jars to Spark



Spark AWS Kinesis Kafka Consumption Model

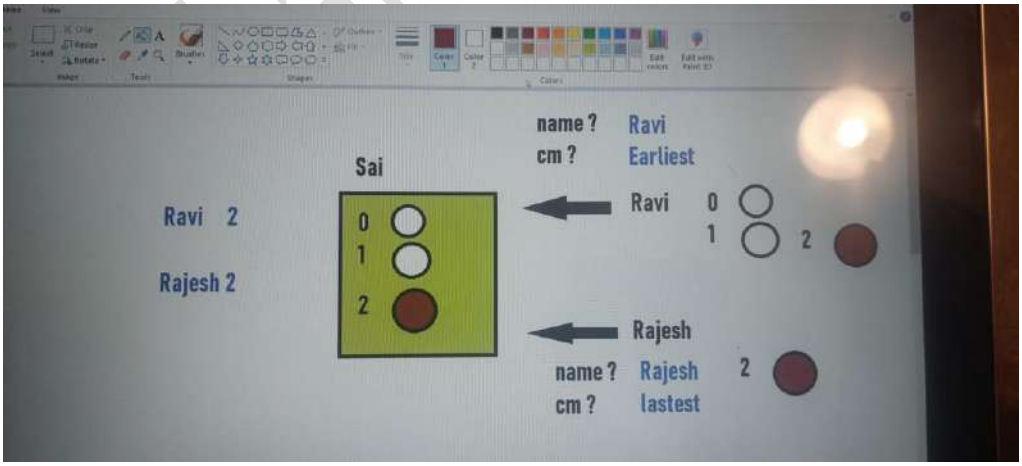
Agenda Day 48

Kafka Consumption model

Kafka by default store messages for 168hrs

Earliest – send all the messages and the upcoming messages.

Latest – Send only upcoming messages



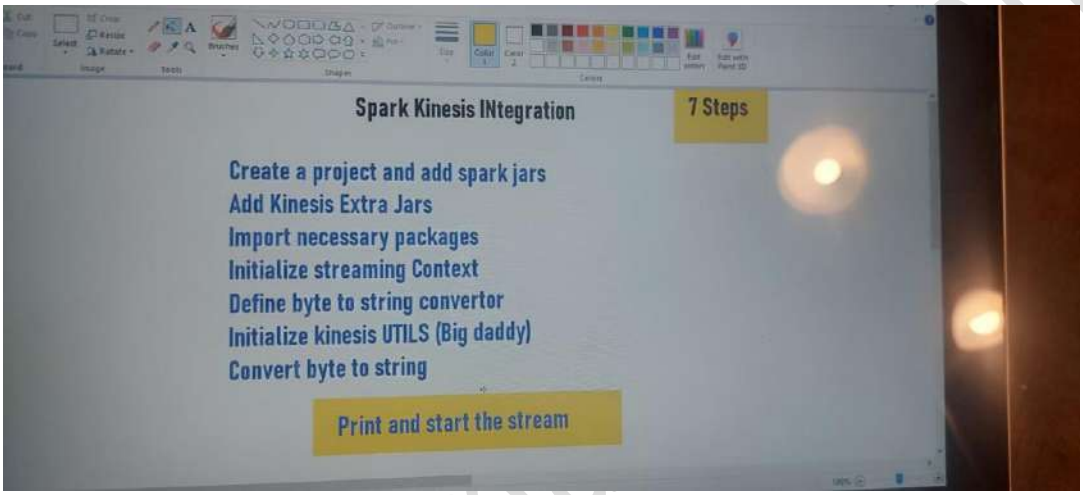
Offset Explorer Download link: https://www.kafkatools.com/download.html

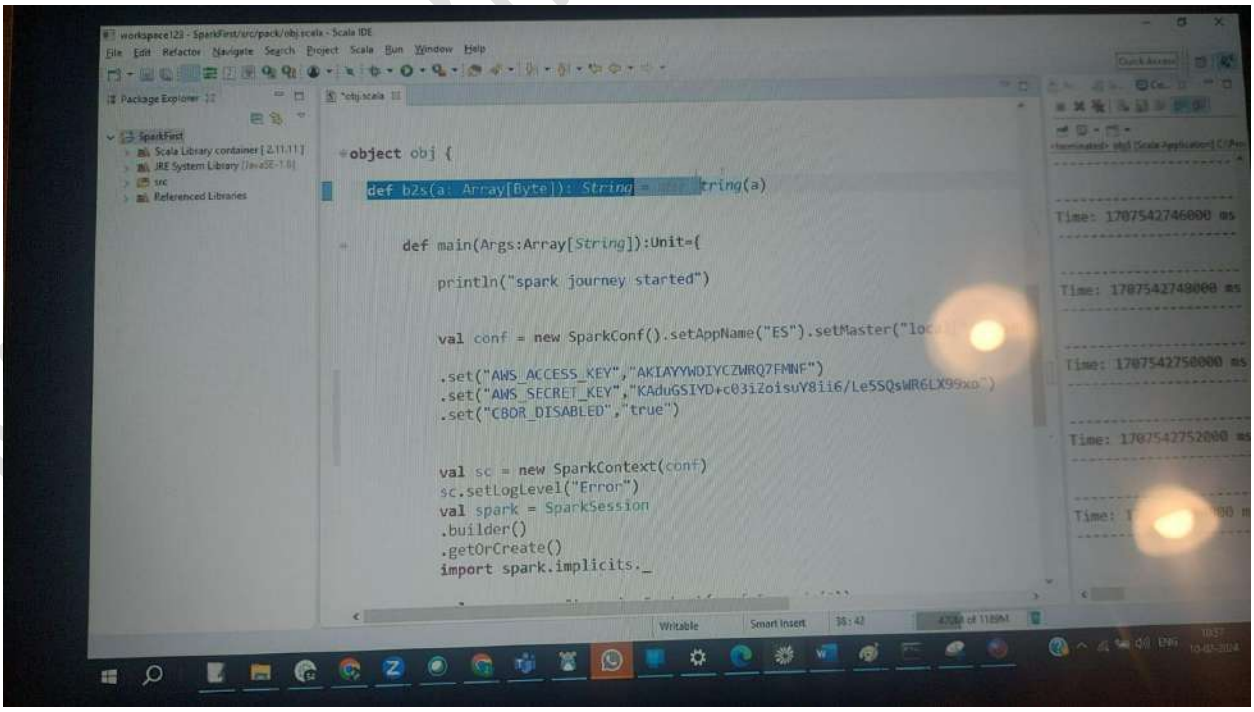
AWS Kinesis, Azure – Event Hub, GCP – PUB/SUB are the MQ in cloud.

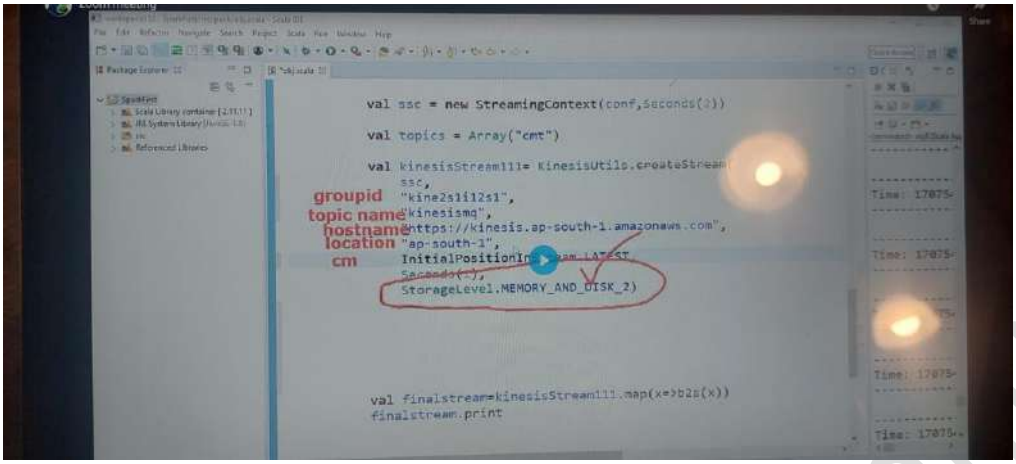
AWS:
Kinesis is a service, DataStream is like topic in kafka.

Steps:

1) Create Data Stream by going to AWS Kinesis
2) Stream Data to Kinesis:







Steps to Start and publish messages to kinesis from CMD: