In [ ]:
```
1.Batch Processing
====================
Process finite data (complete dataset) - stored in DB/File - fetch the complete dat
df = pd.read_csv('emp.csv')
df.shape ->(10,5)
df.shape ->(10,5)
 Vs
2.Stream Processing
====================
Process infinite data (or) real-time data
- read ->transform ->write data

    +-------------------------+
    | Streaming Source <== socket , file, kafka etc
    +-------------------------+
              |
     | SparkStructred Streaming |
     |  (Transformation)        |
              |
     +-------------------------+
     |  Streaming Sink  ---> Console,HDFS,Delta Lake,Kafa etc.,
     +-------------------------+


    DataStream                      Unbound Table
    -----------                    ----------------------
       --------------->              ...
    -----------                    ----------------------
                                     ...
                                   ------------------------
                                      ...
                                    ------------------------

    ---------->| SparkStream | -->[][][][][]  ----->| SparkCore | --> outputSinks
                                   inputbatches
outputmode
->append mode => only new rows - not using aggregate method
->update mode => Incremental value / recente value  - aggregate
->complete mode => Full value / - aggregate


1st spark session object
2nd spark_session_object.interface to inputMode ->df
3rd df.do_Transformation ->results
4th result ->Sinkto_output
```

In [ ]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('demo1').master('local[*]').getOrCreate()

df = spark.readStream.format("socket").option("host","localhost").option("port",112
print(df.isStreaming) # ->bool(True/False)
print(df.printSchema())
write_query = df.writeStream.format("console").start()
write_query.awaitTermination() # keep on running
```

In [ ]:
```
write_query = df.writeStream.format("console").start()
same as
write_query = df.writeStream.outputMode("append").format("console").start()
###
df.select(explode(split(..,' ')))

select()-query/filter
|
+---------+
| value   |
+---------+
| data1 data2 data1
+-------------------+
   |
explode() =>column->row
                    |
                [data1|data2|data1]
```

In [ ]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('demo1').master('local[*]').getOrCreate()

df = spark.readStream.format("socket").option("host","localhost").option("port",112
print(df.isStreaming) # ->bool(True/False)
print(df.printSchema())
write_query = df.writeStream.format("console").start()
write_query.awaitTermination() # keep on running
```

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.appName('demo1').master('local[*]').getOrCreate()

df = spark.readStream.format("socket").option("host","localhost").option("port",112
print(df.isStreaming) # ->bool(True/False)
print(df.printSchema())
###
df_result = df.select(explode(split("value"," ")).alias("word"))

word_count = df_result.groupBy("word").count()
###
#
# write_query = word_count.writeStream.format("console").start()
# Error - default outputMode is append

write_query = word_count.writeStream.outputMode("update").format("console").start()
write_query.awaitTermination() # keep on running
```

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.appName('demo1').master('local[*]').getOrCreate()

df = spark.readStream.format("socket").option("host","localhost").option("port",112
print(df.isStreaming) # ->bool(True/False)
print(df.printSchema())
###
df_result = df.select(explode(split("value"," ")).alias("word"))

word_count = df_result.groupBy("word").count()
###
#
# write_query = word_count.writeStream.format("console").start()
# Error - default outputMode is append

write_query = word_count.writeStream.outputMode("complete").format("console").start
write_query.awaitTermination() # keep on running
```

In [ ]:
```
1.create source directory (ex: input_dir/
                                    - file1.csv
                                    - file2.csv
                                    - ..
                                    - ...
.............................................................
1st =>sparksession_object
2nd =>define schema  <== from pyspark.sql.types import *
          |
         StructType(StructField["pname",StringType(),True],
         StructType(StructField["pname",IntegerType(),True])
        ======================================================
                |->initialize schema object

3rd => read stream from input_dir/
            |
        spark.readStream.option("header","true").schema(<schema object>).csv(input_

4th =>process
|
5th => write streaming to console
```

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField,StringType,IntegerType

spark = SparkSession.builder.appName("filestream").getOrCreate()

schema_obj = StructType([StructField("pname",StringType(),True),StructField("pid",I

file_stream = spark.readStream.option("header","true").schema(schema_obj).csv("inpu

r = file_stream.filter(file_stream.pid >100).groupBy("pname").count()
```

```
r.writeStream.outputMode("complete").format("console").start().awaitTermination()
```

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField,StringType,IntegerType

spark = SparkSession.builder.appName("filestream").getOrCreate()

schema_obj = StructType([StructField("pname",StringType(),True),StructField("pid",I
|
file_stream = spark.readStream.option("header","true").schema(schema_obj).csv("inpu
   Vs
json_stream = spark.readStream.schema(schema_obj).json("json_dir/")
json_stream.writeStream.outputMode("append").format("console").start().awaitTermina
```

In [ ]:
```
mkdir json_dir
cd json_dir
file:p1.py
----------------------------------------
import json
d={'pname':'pA','pid':101}
with open('data1.json','w') as wobj:
    json.dump(d,wobj)
|
python p1.py
|
ls
data1.json <==
==============
--------------------------------------------------------------------------------
Stream from socket ->   ...   ->write to csv file
        ----------    ======     ====================
                        |->append Vs update

input_dir/
    |->data1.csv
    |->data2.csv

output_dir/
    |->result1.csv
        ==========
         |->keep Header line ->option("header","true")

1st create output directory (mkdir output_dir)
..
..
stream_df.writeStream.outputMode("update/append").format("csv").option("path","outp
.option("header","true").start().awaitTermination()
```

In [ ]:
```
# Stream from socket ->process->write to csv file
# ------------------------------------------------------------
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('demo1').master('local[*]').getOrCreate()

df = spark.readStream.format("socket").option("host","localhost")
```

```
.option("port",1120)
.load()

df.writeStream.outputMode("append").format("csv").option("path","output_dir/")
.option("checkpointLocation","checkpoints/csv_stream_chpt")
.option("header","true")
.start().awaitTermination()
```

In [ ]:
```
Hello Good Morining  ->embedding->   [vector]           ..............
                           |->ML         |->vectorDB <-- llm -
(English)       -----------------> French

End User: Get list of sales emp records //plain text/english
          |_____________________________|
                |
                llm
                |
                select *from emp <similarity_search- sales dept ...>
                                  ================================== // SQL
                                      where edept = "sales"  // SQL
                |

from pyspark.ml.feature import VectorAssembler
```

In [ ]:
```
# Stream from socket ->process-> write to csv file
#                       |                       |->update mode
#                       |->do aggregate - word count
# ------------------------------------------------------------
# append mode - wont' support aggregate operation
# ------
# update and complete mode support aggregate operation
# --------------------
# complete mode won't support for file sink - supports console (or) memory
# ========
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.appName('demo1').master('local[*]').getOrCreate()

df = spark.readStream.format("socket").option("host","localhost").option("port",112

df_result = df.select(explode(split("value"," ")).alias("word"))

wc = df_result.groupBy("word").count()
wc.writeStream.outputMode("update").format("csv").option("path","output_dir/")
.option("checkpointLocation","checkpoints/csv_stream_chpt")
.option("header","true").start().awaitTermination()
```

In [ ]:
```
## Apache Flink
------------------
Flink - real-time data processing framework
                ===============
                  |->process streaming data           Structured Stream
API (or) Libs - Flink ML ; Table                Vs     DataFrame (or) DataSet
-----------------------------------------------
  Batch Process    Stream Process
```

```
=============================================
    (Kernel) - Runtime Stream
---------------------------------------------
    Deployment - local ; cluster
-----------------------------------
 Storage - HDFS;DB;..
-----------------------------------------

Realtime Analytics
in E-Commerce platform
[]->    Apache kafka + Flink + ML+ElastricSearch -->Grafana (Visualization)

(Source) -->Flink        -------->[Sink]
              -Operators
               map,filter,flatMap,keyby.
    Operators like builtin function


Appln
 <------>DB
 ------>GeneratesLog (events)
                |---------------------+Flink+Elaststic(or)Loki-->Grafana

pip install apache-flink
|
pyflink/  <== from pyflink.datastream import StreamExecutionEnvironment
    |
    Set up -env   env=StreamExecutionEnvironment.get_environment()
    |
    read   env.socket_text_stream("node",<port>) ->data_stream
    |
    transformation    results=data_stream.flat_map(...)
    |
    sinks              results.add_sink(sink_obj)
```

In [ ]:
```
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.common.typeinfo import Types
env = StreamExecutionEnvironment.get_execution_environment()
env.set_parallelism(1)
data = ["hello data","hello flink","flink data streaming"]
text_stream = env.from_collection(data,type_info=Types.STRING())

wc = (text_stream.flat_map(lambda line:[(w,1) for w in line.split()],output_type=Ty
wc.print()
env.execute("simple wc")
```

In [ ]:
```
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.common.typeinfo import Types
from pyflink.datastream.connectors import StreamingFileSink

#from pyflink.datastream.formats import CsvEncoder - Version 1.6 used

from pyflink.common.serialization import Encoder
import os
```

```
class CsvEncoder(Encoder):
    def __init__(self):
        super().__init__(self)
    def encoder(self,value,stream):
        line = ",".join(map(str,value))+"\n"
        stream.write(line.encode("utf-8"))

env = StreamExecutionEnvironment.get_execution_environment()
env.set_parallelism(1)
#####################################################

data = ["hello stream","hello flink","flink streaming"]
text_stream = env.from_collection(data,type_info=Types.STRING())
#####################################################

wc = (text_stream.flat_map(lambda a:[(w,1) for w in a.split()],output_type=Types.TU
#####################################################
var="output_csv_dir"
os.makedirs(var,exist_ok=True) # create output directory

csv_sink = StreamingFileSink.for_row_format(var,CsvEncoder()).build()

wc.add_sink(csv_sink) # Write results to csv

#####################################################
env.execute("Socket Stream to csvfile")
# submits the job to Flink runtime and starts execution
```

In [ ]:

```
Linux -> crontab - cron  ->crond //os - jobscheduling
|
Apache airflow
================
|->Open-Source tool - data pipeline //code

DAG
---
download_webcontent  <== t1
extract data  <== t2
|
DataFrame <== t3
|
Insert to DB <--t4

1. scheduler
2. executor - task
3. webUI
4. Operator - predefined templates/scripts

On Linux: python p1.py <==
     |
    vi p1.sh
    python p1.py <==
    :wq
    chmod +x p1.sh
```

```
    ./p1.sh <== running shellscript -- python code executed by bash

 5. scheduler
      - job1
      - job2
      - job3
        ..//queue
 6. worker - instance
 7. metadata
     -- DAG run,status,log ...//

airflow/
     |->logs/
     |->data/
     |->dags/ <==
          |
          |<-- job_schedule_task_in_python_code_Style //dagscript.py
|
After schedule this task
|
Start Webserver and Scheduler
 webserver => airflow webserver --port <portNumber>
 scheduler => airflow scheduler
|
Go to broswer => http://localhost:<port>{enter}

        Login :  _____ <== airflow login
        password: ____ <== airflow password
    |
  [DAG]
    |-> dagName/id
-----------------------------------------------------------------------
DAG_Script_template/format
|
from airflow import DAG
from airflow.operators.python import PythonOperator
|
ContextSwitch => with DAG(<params>) as dag_obj:

<params>
   |->dag_id=<userdefined dagName>
      start_date=datetime(start up date) # YYYY,MM,DD
      schedule_interval='@daily' 5minutes '@weekly'
      catchup=<bool>
               |->True - 2025,10,15 - scheduled date
                                    |//pending jobs - run the pending jobs
                        started on 17th oct

     PythonOperator(task_id=<>,python_callable=<functionName>)
     BashOperator()

         job
    +-----------+
    |           |
    +-----------+
```

In [ ]:
```
#python3 -m venv airflow_project
#source airflow_project/bin/activate
(airflow_project) #
(airflow_project) #export AIRFLOW_VERSION=2.10.1
(airflow_project) #export PYTHON_VERSION="$(python --version|cut -d" " -f 2|cut -d
(airflow_project) #export CONSTRAINT_URL="https://raw.githubusercontent.com/apache/
(airflow_project) #
(airflow_project) #pip install "apache-airflow == ${AIRFLOW_VERSION}" --constraint
(airflow_project) #airflow version
2.10.1
(airflow_project) #airflow db init
(airflow_project) # To create airflow login and password
airflow users create --username admin --firstname student --lastname user --role ad
|
| To start webserver
(airflow_project) #airflow webserver -p 8080

open another terminal => Activate env =>
                Start scheduler => airflow_project) student@paka:~/airflow-Demo$ ai


   |
open a webbroswer => localhost:8080 {Enter}

                   Login: admin
                   Password: admin
                       |
```

In [ ]:
```
(airflow_project) student@paka:~$ ls airflow
airflow-webserver.pid  airflow.cfg  airflow.db  logs  webserver_config.py
                            ------------------------------//there is no dags direct
(airflow_project) student@paka:~$ mkdir -p ~/airflow/dags <== Create new dags direc
(airflow_project) student@paka:~$ ls airflow
airflow-webserver.pid  airflow.cfg  airflow.db  dags  logs  webserver_config.py
(airflow_project) student@paka:~$              ------
|
Copy our dagscript.py file to ~/airflow/dags/
-----------------------------------------------
|
Restart webserver and scheduler
|
open broswer ->127.0.0.1:8080 =>login: __  password:__
================================================================================
```