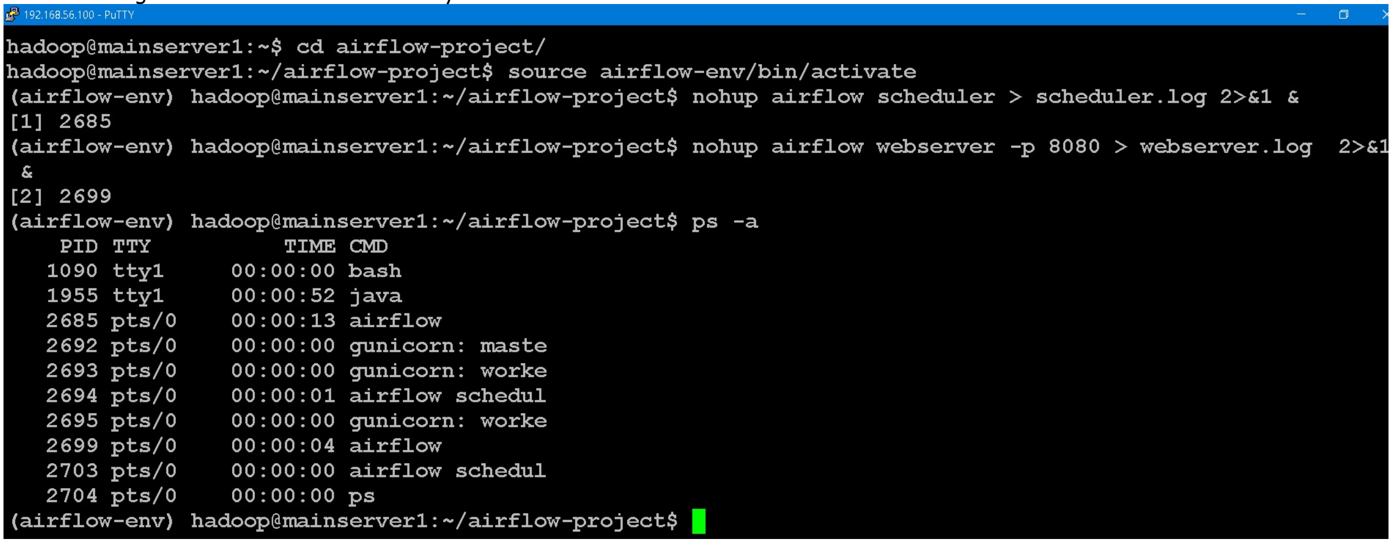
BigData Concepts

09 January 2024 09:10

Airflow final example to demo ETL

Python Script that copies data from local filesystem to HDFS and an Apache Airflow DAG to orchestrate the workflow.

Before we begin - the current status of the systems



```
hadoop@mainserver1:~$ cd airflow-project/
hadoop@mainserver1:~/airflow-project$ source airflow-env/bin/activate
(airflow-env) hadoop@mainserver1:~/airflow-project$ nohup airflow scheduler > scheduler.log 2>&1 &
[1] 2685
(airflow-env) hadoop@mainserver1:~/airflow-project$ nohup airflow webserver -p 8080 > webserver.log  2>&1
 &
[2] 2699
(airflow-env) hadoop@mainserver1:~/airflow-project$ ps -a
    PID TTY          TIME CMD
   1090 tty1     00:00:00 bash
   1955 tty1     00:00:52 java
   2685 pts/0    00:00:13 airflow
   2692 pts/0    00:00:00 gunicorn: maste
   2693 pts/0    00:00:00 gunicorn: worke
   2694 pts/0    00:00:01 airflow schedul
   2695 pts/0    00:00:00 gunicorn: worke
   2699 pts/0    00:00:04 airflow
   2703 pts/0    00:00:00 airflow schedul
   2704 pts/0    00:00:00 ps
(airflow-env) hadoop@mainserver1:~/airflow-project$
```

Create a python script as below

```
$ vim local_to_hdfs.py
```

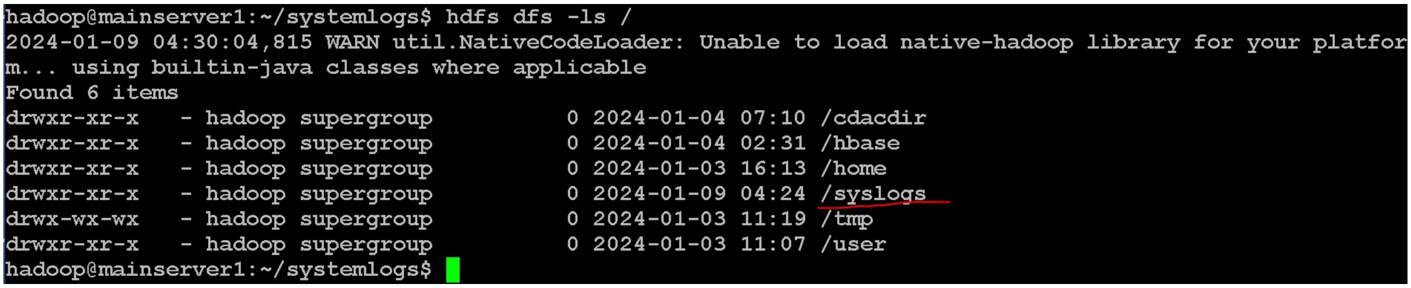
```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.providers.http.sensors.http import HttpSensor
from airflow.sensors.filesystem import FileSensor
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator

default_args ={
    "owner" :"airflow",
    "email_on_failure" : False,
    "email_on_retry" : False,
    "email": "your email",
    "retries" :1,
    "retry_delay" : timedelta(minutes=5)
}

with DAG("Copy_To_HDFS",start_date=datetime(2024,1,8),
    schedule_interval="@daily",default_args=default_args, catchup=False) as dag:
    push_file_hdfs=BashOperator(
        task_id="push_file_hdfs",
        bash_command= """
            hdfs dfs -mkdir -p /syslogs/$HOSTNAME && \
            hdfs dfs -put -f $HOME/systemlogs/*  /syslogs/$HOSTNAME
        """
)
```

<save and exit>

```
$ hadoop  dfs -ls  /
```

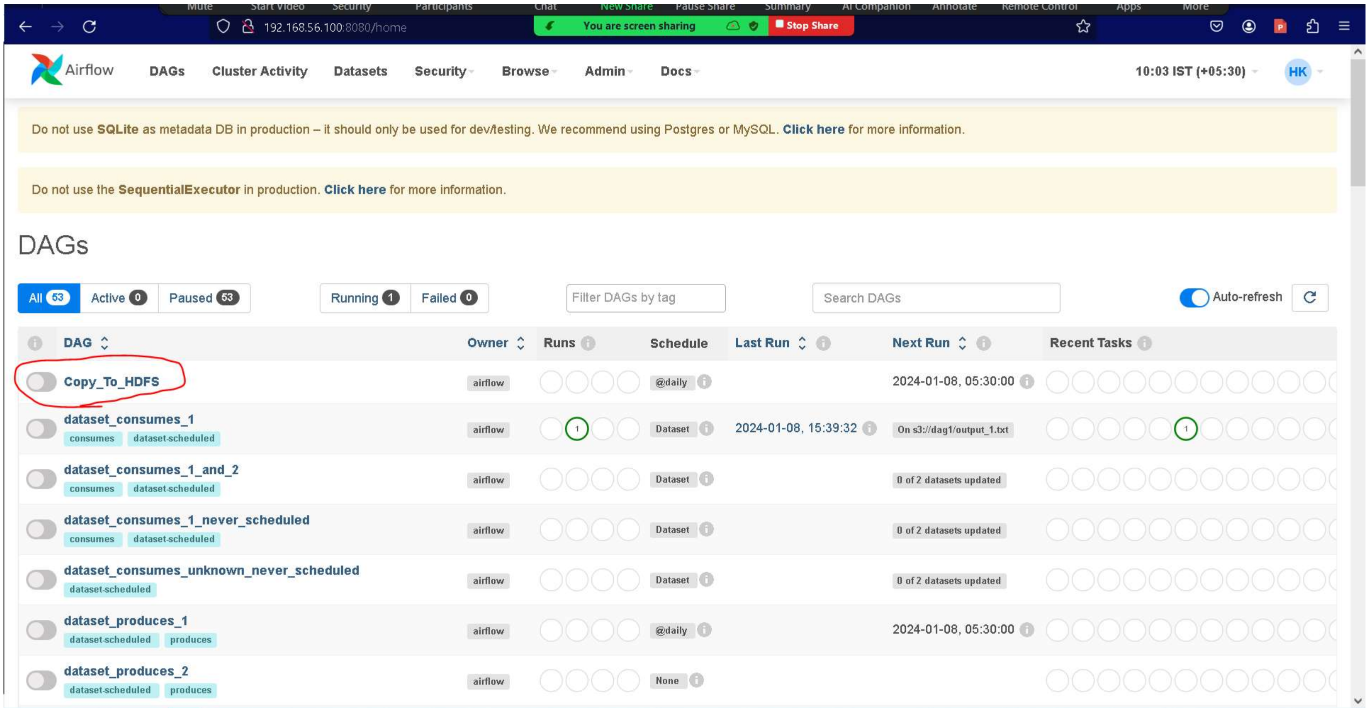
Verify if syslogs directory is present in the outout if now use the below command to create one

```
$ hadoop dfs -mkdir /syslogs
```



```
hadoop@mainserver1:~/systemlogs$ hdfs dfs -ls /
2024-01-09 04:30:04,815 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platfor
m... using builtin-java classes where applicable
Found 6 items
drwxr-xr-x   - hadoop supergroup          0 2024-01-04 07:10 /cdacdir
drwxr-xr-x   - hadoop supergroup          0 2024-01-04 02:31 /hbase
drwxr-xr-x   - hadoop supergroup          0 2024-01-03 16:13 /home
drwxr-xr-x   - hadoop supergroup          0 2024-01-09 04:24 /syslogs
drwx-wx-wx   - hadoop supergroup          0 2024-01-03 11:19 /tmp
drwxr-xr-x   - hadoop supergroup          0 2024-01-03 11:07 /user
hadoop@mainserver1:~/systemlogs$
```

Now go the airflow web and identify the DAG which you have created.



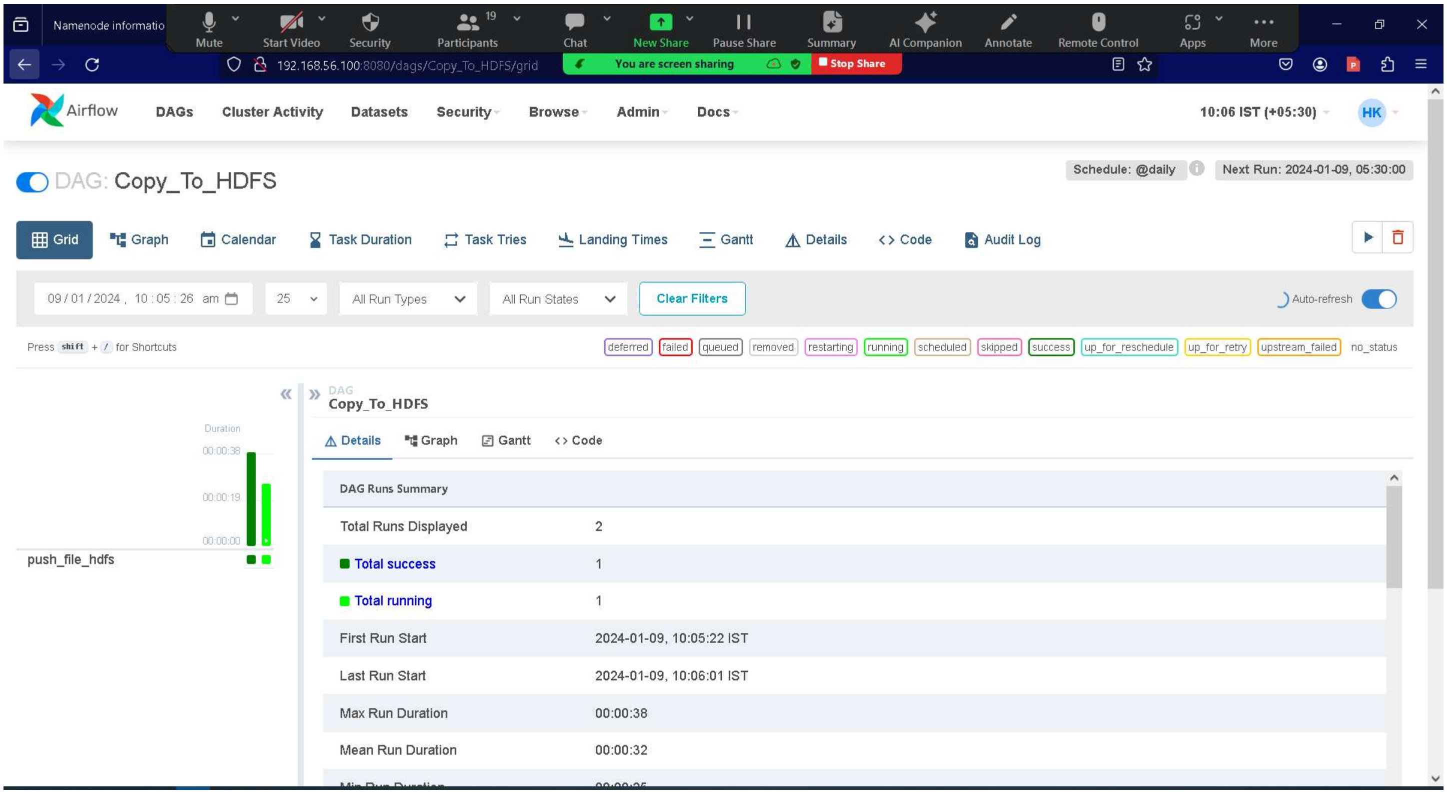
Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. **Click here** for more information.

Do not use the **SequentialExecutor** in production. **Click here** for more information.

DAGs

All 53 | Active 0 | Paused 53 | Running 1 | Failed 0 | Filter DAGs by tag | Search DAGs | Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
Copy_To_HDFS	airflow		@daily		2024-01-08, 05:30:00	
dataset_consumes_1 consumes dataset-scheduled	airflow	1	Dataset	2024-01-08, 15:39:32	On s3://dag1/output_1.txt	1
dataset_consumes_1_and_2 consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated	
dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated	
dataset_consumes_unknown_never_scheduled dataset-scheduled	airflow		Dataset		0 of 2 datasets updated	
dataset_produces_1 dataset-scheduled produces	airflow		@daily		2024-01-08, 05:30:00	
dataset_produces_2 dataset-scheduled produces	airflow		None			



DAG: Copy_To_HDFS

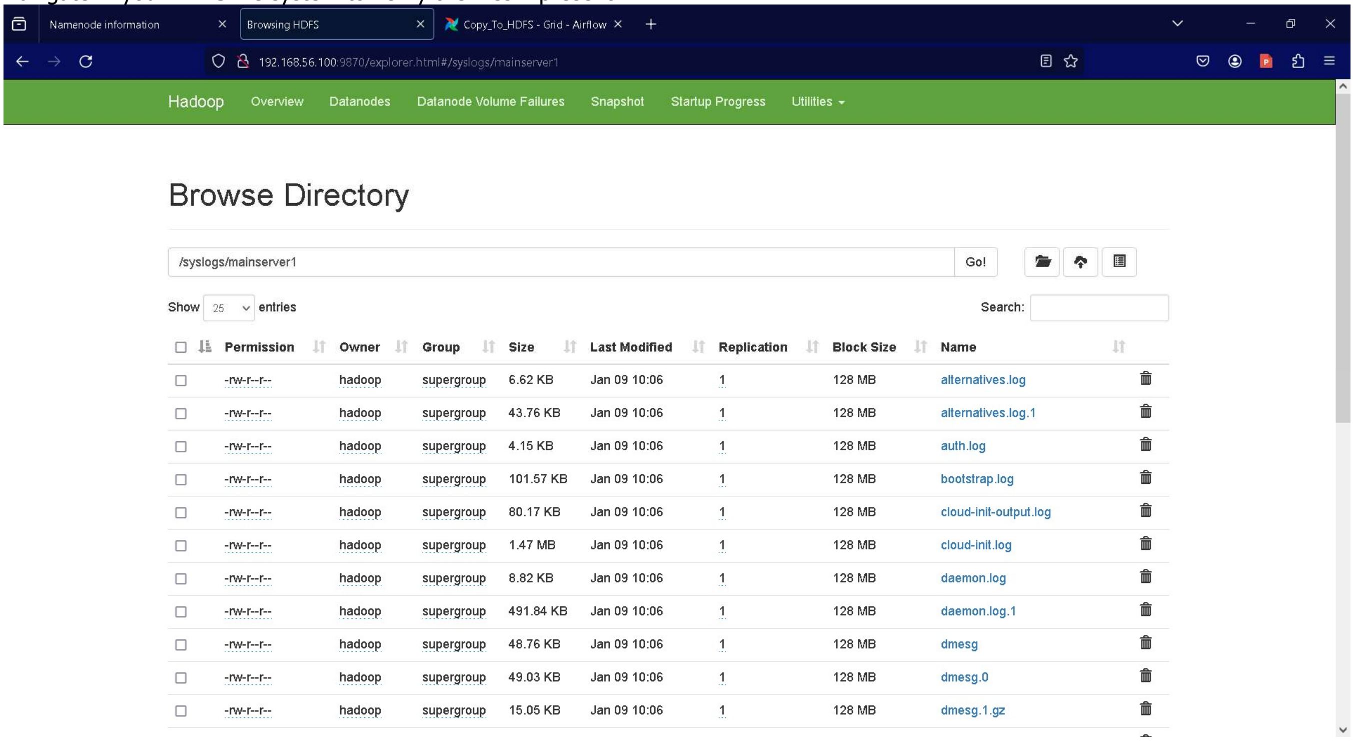
Schedule: @daily | Next Run: 2024-01-09, 05:30:00

09 / 01 / 2024 , 10 : 05 : 26 am | 25 | All Run Types | All Run States | Clear Filters | Auto-refresh

Press shift + / for Shortcuts

DAG Runs Summary	
Total Runs Displayed	2
🟩 Total success	1
🟩 Total running	1
First Run Start	2024-01-09, 10:05:22 IST
Last Run Start	2024-01-09, 10:06:01 IST
Max Run Duration	00:00:38
Mean Run Duration	00:00:32
Min Run Duration	00:00:25

Navigate in your HDFS file system to verify the files if present



Browse Directory

/syslogs/mainserver1

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	hadoop	supergroup	6.62 KB	Jan 09 10:06	1	128 MB	alternatives.log
□	-rw-r--r--	hadoop	supergroup	43.76 KB	Jan 09 10:06	1	128 MB	alternatives.log.1
□	-rw-r--r--	hadoop	supergroup	4.15 KB	Jan 09 10:06	1	128 MB	auth.log
□	-rw-r--r--	hadoop	supergroup	101.57 KB	Jan 09 10:06	1	128 MB	bootstrap.log
□	-rw-r--r--	hadoop	supergroup	80.17 KB	Jan 09 10:06	1	128 MB	cloud-init-output.log
□	-rw-r--r--	hadoop	supergroup	1.47 MB	Jan 09 10:06	1	128 MB	cloud-init.log
□	-rw-r--r--	hadoop	supergroup	8.82 KB	Jan 09 10:06	1	128 MB	daemon.log
□	-rw-r--r--	hadoop	supergroup	491.84 KB	Jan 09 10:06	1	128 MB	daemon.log.1
□	-rw-r--r--	hadoop	supergroup	48.76 KB	Jan 09 10:06	1	128 MB	dmesg
□	-rw-r--r--	hadoop	supergroup	49.03 KB	Jan 09 10:06	1	128 MB	dmesg.0
□	-rw-r--r--	hadoop	supergroup	15.05 KB	Jan 09 10:06	1	128 MB	dmesg.1.gz

12. Introduction to Apache Spark

- Apache Spark APIs for large-scale data processing

Lecture

- Overview, Linking with Spark, Initializing Spark,
- Resilient Distributed Datasets (RDDs), External Datasets
- RDD v/s Data frames v/s Datasets
- Data frame operations
- Structured Spark Streaming
- Passing Functions to Spark, Working with Key-Value Pairs, Shuffle operations,
- RDD Persistence, Removing Data, Shared Variables, Deploying to a Cluster

Lab-Assignment:

- Run the provided Hadoop Streaming program using python

Lecture

- Map Reduce with Spark
- Working with Spark with Hadoop
- Working with Spark without Hadoop and their Differences

Lab Assignment

- Execute all the provided code using step-runs for each and every codeline
- Setup the JDBC configuration and run the Spark JDBC Connectivity program
- Run the spark integrations using the provided code

Lecture

- Data preprocessing
- EDA

Introduction to Apache Spark

Apache Spark is an open-source, distributed computing system that provides a fast and general-purpose cluster computing framework for big data processing.
It was developed to address the limitations of the MapReduce model, providing a more flexible and efficient alternative for large-scale data processing.

Its primary purpose is to handle the real-time generated data.

Spark was built on the top of the Hadoop MapReduce.

It was optimized to run in memory whereas alternative approaches like Hadoop's MapReduce writes data to and from computer hard drives. So, Spark process the data much quicker than other alternatives.

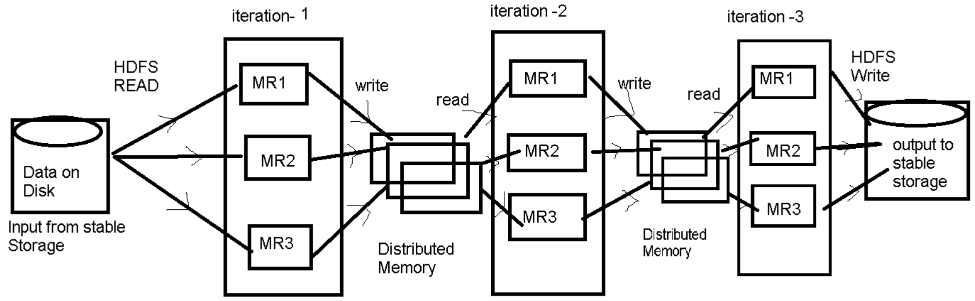
History :- The Spark was initiated by Matei Zaharia at UC Berkeley's AMP Lab in 2009. It was open sourced in 2010 under a BSD license. In 2013, the project was acquired by Apache Software Foundation. In 2014, the Spark emerged as a Top-Level Apache Project.

- Faster, Easy to use (Java , Scala, Python, R and SQL), lightweight, Runs Everywhere (you can run it on Hadoop, Standalone, on the cloud, or kubernetes)
- Provides high performance for both batch and streaming data, using DAG Schedulers, a query optimizer and a physical execution engine.

Usage case : Data integration, stream processing, Machine learning (since spark is capable of storing data in memory and can run repeated queries quickly, hence ml algorithms can work on this data easily), interactive analytics.

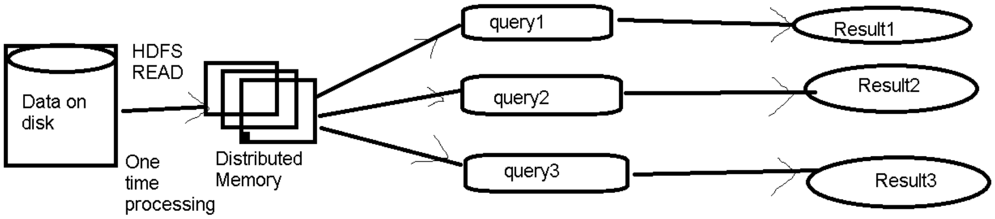
Key Features of Apache Spark:

1. **Resilient Distributed Datasets (RDS):**



Iterative Operations on Sparck RDD

- IT will store intermediate results in a distributed memory instead of stable storage (disk) and hence makes the systems faster.
- If the distributed memory (RAM) is not sufficient to store intermediate results (state of the JOB), then it will store those results on the disk.



- If different queries are un on the same set of data repeatedly, this data (repeatedly queried data) can be kept in memory for better execution times.

Cache Terminologies :
- Cache hit --> Data present in the cache memory (read cache hit)
- Cache miss --> Data is not present in the cache memory and we need to fetch the information from the original location (read cache miss)
- Cache dirty / Dirty Cache --> Data present in the cache but not yet committed to the disk ie., original locations
- Write through cache --> data is first written to cache and then to the disk after it is committed to the original locations acknowledge is set the application
- Write back cache --> data is first written to cache and acknowledge is send the application immediately , but later on t committed to the original location
- *Cache Flush -> clear all you data present in the cache
- *Cache valut --> safe area where you try to store the cache data safely till the operations returns to the normal state.

By default each transformed RDD may be recomputed each time we run an action on it but we can also **"persist"** an RDD in memory, in this case spark will keep the elements around on the cluster for faster access, next time when we query.
Supports for Persisting RDD's on disks or replicated across multiple nodes.

In case of data sharing in MapReduce is slower because of replication, serialization and disk IO. Most of the Hadoop apps, time is utilized in performing rea-write operations (almost 90%). Hence we are using RDD to overcome this issues. Because RDD supports in-memory processing computation.

Components of Apache Spark --> RDD's, Spark Core, Spark SQL, spark streaming, Mlib (Machine Learning Library), GraphX, SparkR,Sparks Cluster Manager.

Spark Core ---> Basic function --> task scheduling, memory management, fault recovery and integration and interaction with storage systems.
Spark Cluster Manager --> Hadoop Yarn, Apache Mesos, and its standalone cluster manager.
Spark SQL --> Allows used to execute SQL - like queries

Note: Spark is not a modified version of Hadoop and not really dependent on Hadoop because it has it won cluster management. But Spark uses Hadoop for storage and processing (in memory).

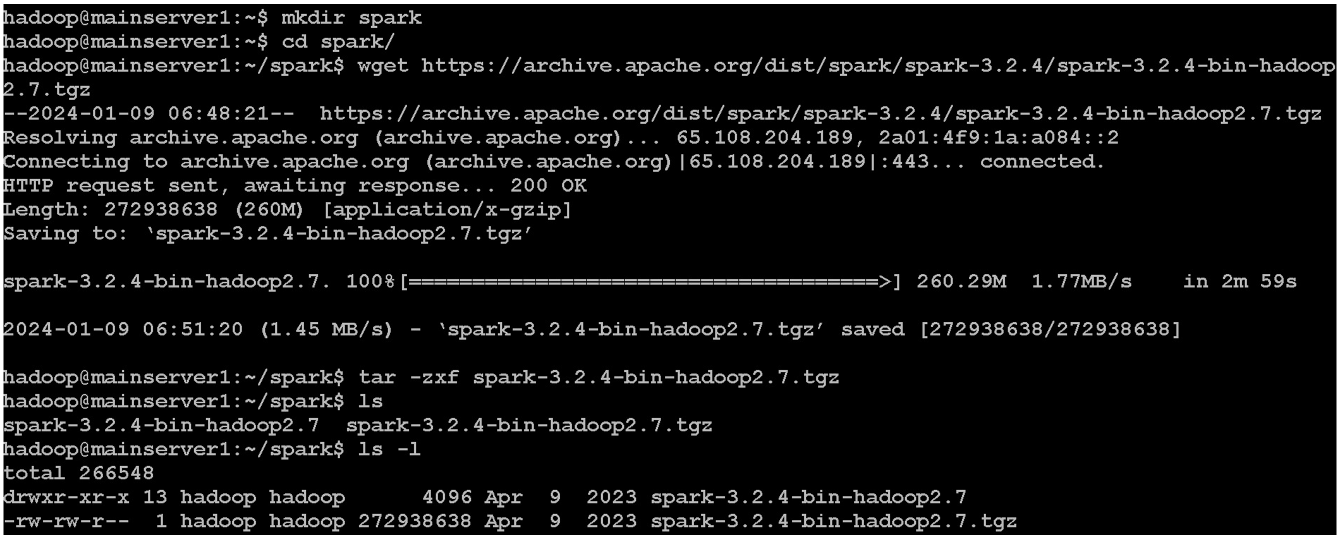
Deployment of Spark

- Standalone --> HDFS and Spark
- Hadoop Yarn ---> HDFS , YARN or Mesos , Spark (applicable in Hadoop 2.x and above)
- Spark in MapReduce (SIMR) --> HDFS, MapReduce, Spark (applicable in Hadoop 1.x)

Apache Spark Installation:

As per the current lab setup , java, HDFS, and yarn is already working perfectly.

$ mkdir spark
$ cd spark
$ wget https://archive.apache.org/dist/spark/spark-3.2.4/spark-3.2.4-bin-hadoop2.7.tgz
$ tar -zxf spark-3.2.4-bin-hadoop2.7.tgz

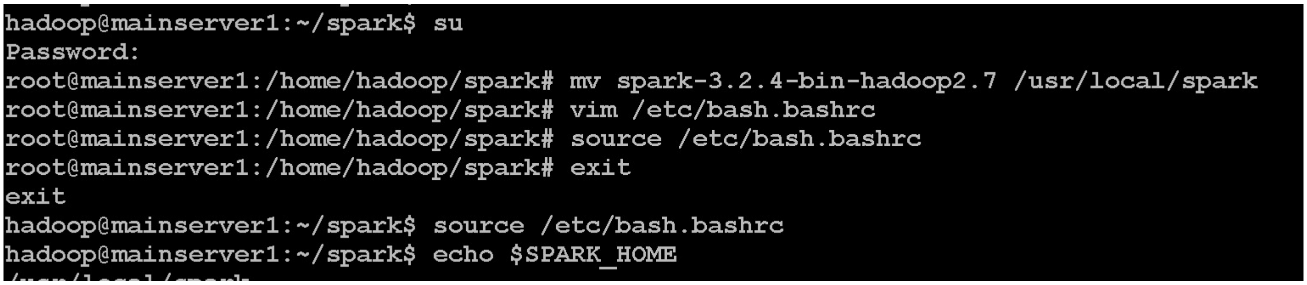


```
hadoop@mainserver1:~$ mkdir spark
hadoop@mainserver1:~$ cd spark/
hadoop@mainserver1:~/spark$ wget https://archive.apache.org/dist/spark/spark-3.2.4/spark-3.2.4-bin-hadoop
2.7.tgz
--2024-01-09 06:48:21--  https://archive.apache.org/dist/spark/spark-3.2.4/spark-3.2.4-bin-hadoop2.7.tgz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189, 2a01:4f9:1a:a084::2
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 272938638 (260M) [application/x-gzip]
Saving to: 'spark-3.2.4-bin-hadoop2.7.tgz'

spark-3.2.4-bin-hadoop2.7. 100%[=====================================>] 260.29M  1.77MB/s    in 2m 59s

2024-01-09 06:51:20 (1.45 MB/s) - 'spark-3.2.4-bin-hadoop2.7.tgz' saved [272938638/272938638]

hadoop@mainserver1:~/spark$ tar -zxf spark-3.2.4-bin-hadoop2.7.tgz
hadoop@mainserver1:~/spark$ ls
spark-3.2.4-bin-hadoop2.7  spark-3.2.4-bin-hadoop2.7.tgz
hadoop@mainserver1:~/spark$ ls -l
total 266548
drwxr-xr-x 13 hadoop hadoop      4096 Apr  9  2023 spark-3.2.4-bin-hadoop2.7
-rw-rw-r--  1 hadoop hadoop 272938638 Apr  9  2023 spark-3.2.4-bin-hadoop2.7.tgz
```

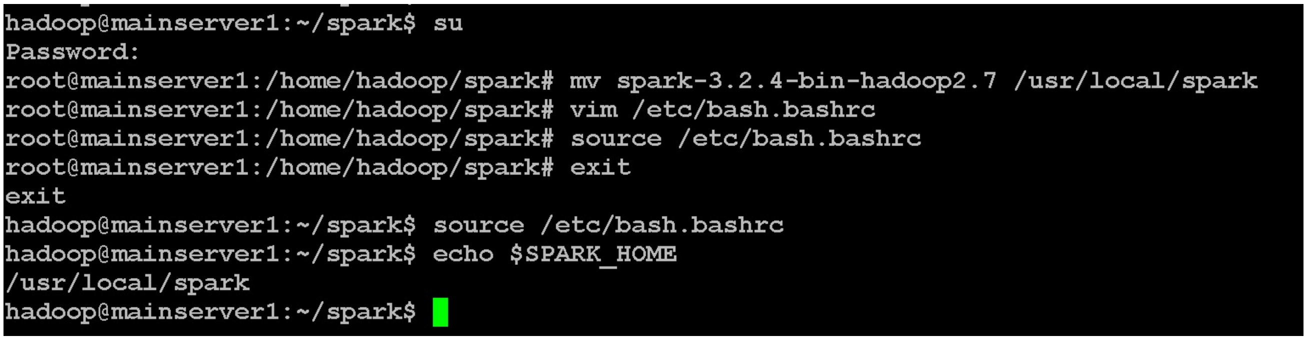
$ su -
# mv
# nano /etc/bash.bashrc
{ Add the following lines in this file }

# APACHE SPARK CONFIG
SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin

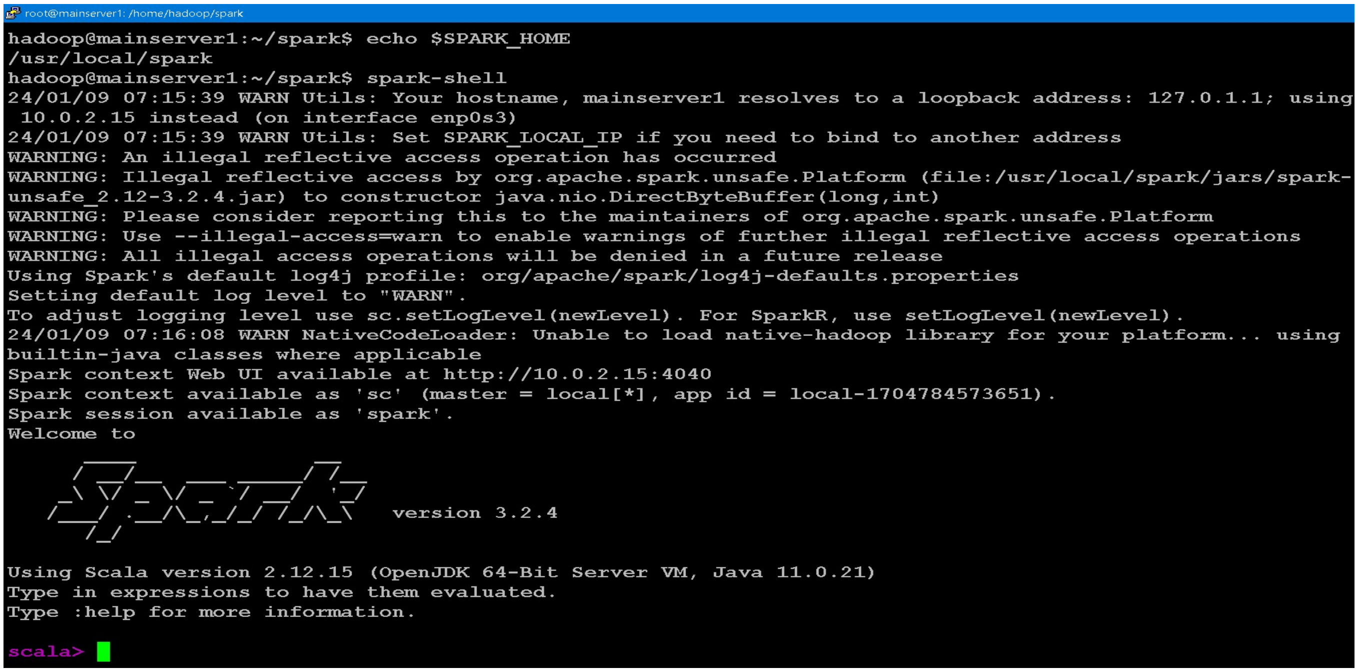
<save and exit>
# exit
$ source /etc/bash.bashrc
$ echo $SPARK_HOME
$ spark-shell



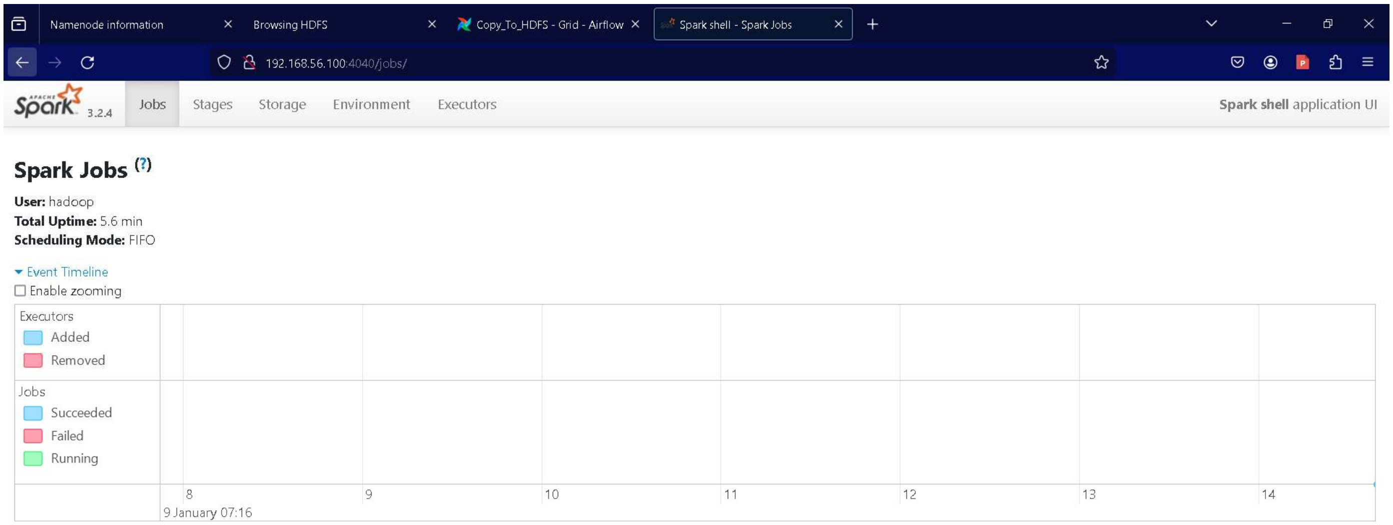
```
hadoop@mainserver1:~/spark$ su
Password:
root@mainserver1:/home/hadoop/spark# mv spark-3.2.4-bin-hadoop2.7 /usr/local/spark
root@mainserver1:/home/hadoop/spark# vim /etc/bash.bashrc
root@mainserver1:/home/hadoop/spark# source /etc/bash.bashrc
root@mainserver1:/home/hadoop/spark# exit
exit
hadoop@mainserver1:~/spark$ source /etc/bash.bashrc
hadoop@mainserver1:~/spark$ echo $SPARK_HOME
/usr/local/spark
```



```
hadoop@mainserver1:~/spark$ su
Password:
root@mainserver1:/home/hadoop/spark# mv spark-3.2.4-bin-hadoop2.7 /usr/local/spark
root@mainserver1:/home/hadoop/spark# vim /etc/bash.bashrc
root@mainserver1:/home/hadoop/spark# source /etc/bash.bashrc
root@mainserver1:/home/hadoop/spark# exit
exit
hadoop@mainserver1:~/spark$ source /etc/bash.bashrc
hadoop@mainserver1:~/spark$ echo $SPARK_HOME
/usr/local/spark
hadoop@mainserver1:~/spark$ 
```



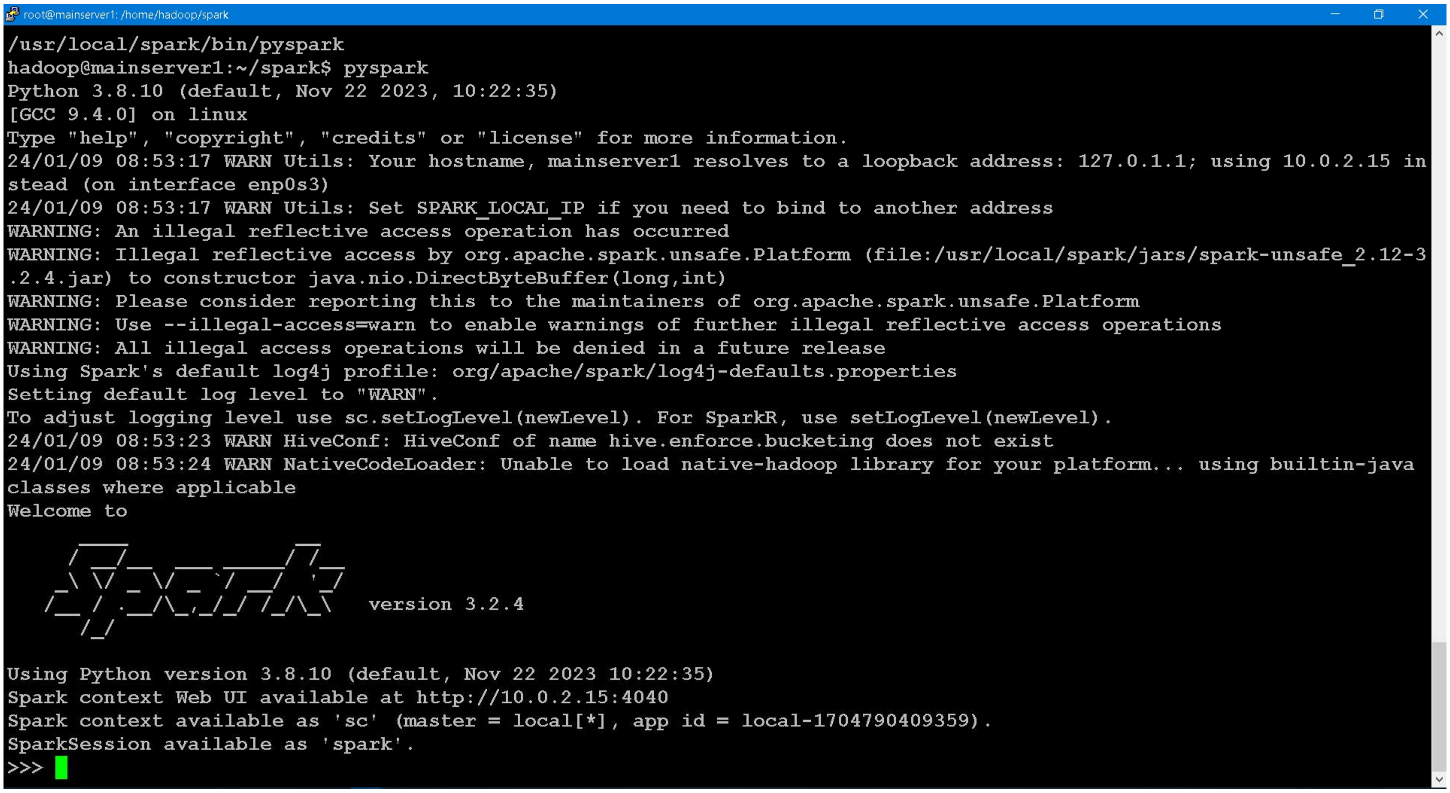
```
hadoop@mainserver1:~/spark$ echo $SPARK_HOME
/usr/local/spark
hadoop@mainserver1:~/spark$ spark-shell
24/01/09 07:15:39 WARN Utils: Your hostname, mainserver1 resolves to a loopback address: 127.0.1.1; using
 10.0.2.15 instead (on interface enp0s3)
24/01/09 07:15:39 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/spark/jars/spark-
unsafe_2.12-3.2.4.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/01/09 07:16:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1704784573651).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.2.4
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.21)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```



Spark Jobs (?)

User: hadoop
Total Uptime: 5.6 min
Scheduling Mode: FIFO

▾ Event Timeline
□ Enable zooming



```
/usr/local/spark/bin/pyspark
hadoop@mainserver1:~/spark$ pyspark
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
24/01/09 08:53:17 WARN Utils: Your hostname, mainserver1 resolves to a loopback address: 127.0.1.1; using 10.0.2.15 in
stead (on interface enp0s3)
24/01/09 08:53:17 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/spark/jars/spark-unsafe_2.12-3
.2.4.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/01/09 08:53:23 WARN HiveConf: HiveConf of name hive.enforce.bucketing does not exist
24/01/09 08:53:24 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.2.4
      /_/

Using Python version 3.8.10 (default, Nov 22 2023 10:22:35)
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1704790409359).
SparkSession available as 'spark'.
>>>
```

Spark Architecture

Spark follows the master-slave architecture. Its cluster consists of single-master and multiple slaves.
The architecture depends on two components / abstractions

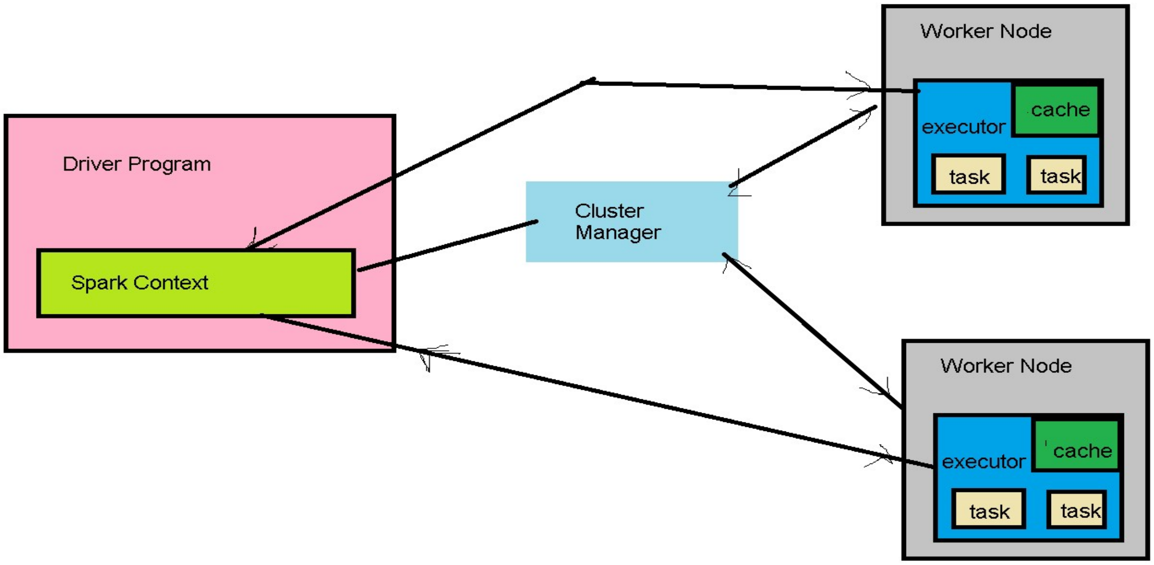
- Resilient Distributed Dataset (RDD)
- Directed Acyclic Graph (DAG)

Resilient Distributed Dataset (RDD)

- It is a group of data items that can be store in-memory on worker nodes
  - Resilient --> restore the data on failure
  - Distributed --> data is distributed among different nodes
  - Datasets --> group of data

Directed Acyclic Graph

- It is a finite direct graph that performs a sequence of computations on data
- Each node is an RDD partition and the edge is a transformation on top of data



Driver Program

- It is a process that runs the Main() functions of the applications and creates the SparkContext object.
- SprakContext - coordinates the spark applications, running as independent sets of processes on a cluster

Inorder to run on a cluster the sparkcontext connects to a different type of cluster managers and then performs the following tasks

- It acquires executors on nodes in the cluster
- Then it sends the applications codes (your app code) to the executors, here the application code can be defined by JAR or Python file passed to the sparkcontext
- At last the spark context sends task to the executors to run

Cluster Manger

- The role of the cluster manager is to allocate resources across applications. { spark is capable of running on a large number of clusters }
- It consists of various types of cluster mangers such as Hadoop Yarn , Apache Mesos and Standalone scheduler
- The standalone scheduler is a standalone spark cluster manager that facilitates us to install spark on an empty set of systems / machines

Worker Nodes

- The worker node is a slave node
- Its role is to run the application code in the cluster

Executor

- It is a process launched for an application on a worker node
- It runs task and keeps data in memory or disk storage across them
- It read and write data to the external sources
- Every applications contains its executors

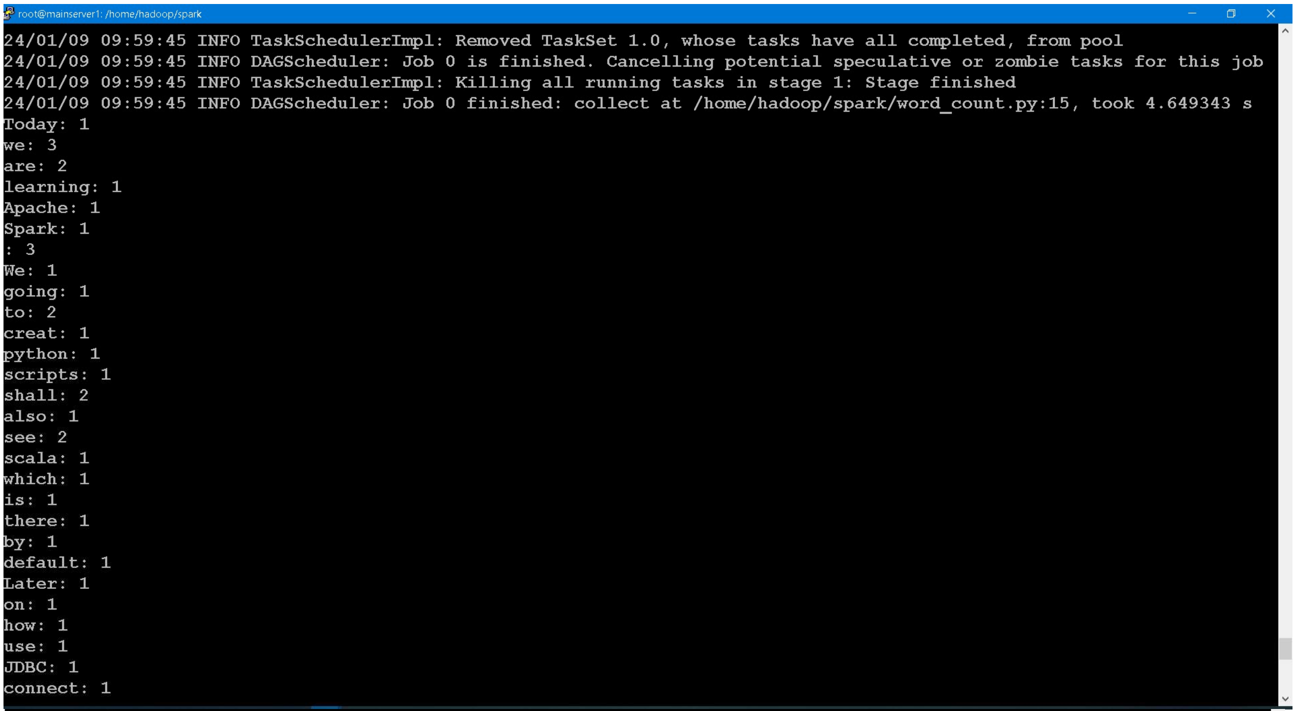
Task

- A unit of work that will be sent to one executor

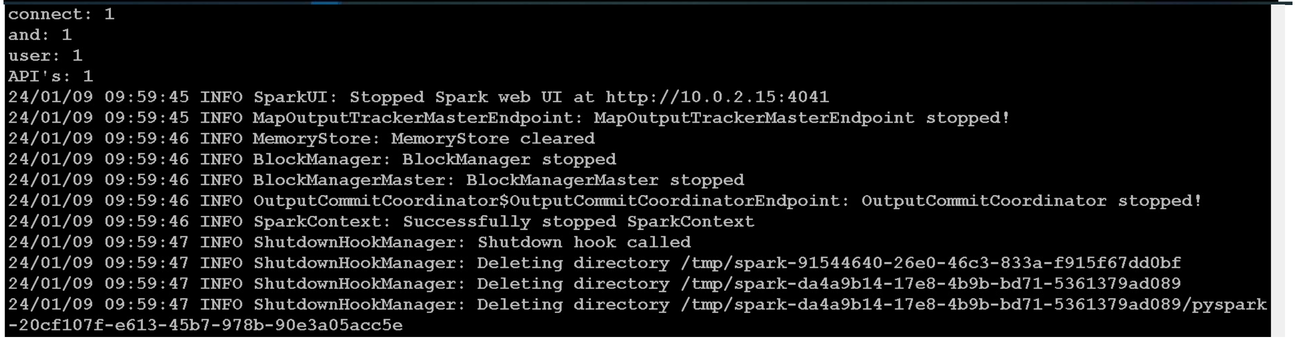
Sample Word_Count program in python

```
$ vim mylocal.txt
        Today we are learning Apache Spark

        We are going to creat python scripts
        we shall also see scala which is there by default

        Later on we shall see how to use JDBC connect
        and user API's
<Save and exit>

$ vim word_count.py
        from pyspark import SparkContext

        # Create a SparkContext
        sc = SparkContext("local", "WordCount")

        # Load a text file
        text_file = sc.textFile("/home/hadoop/spark/mylocal.txt")

        # Perform word count
        word_counts = text_file.flatMap(lambda line: line.split(" ")) \
                          .map(lambda word: (word, 1)) \
                          .reduceByKey(lambda a, b: a + b)

        # Display the result
        for word, count in word_counts.collect():
            print(f"{word}: {count}")

        # Stop the SparkContext
        sc.stop()
        <save and exit>

$  spark-submit word_count.py
```

Output Sample



```
24/01/09 09:59:45 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
24/01/09 09:59:45 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
24/01/09 09:59:45 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
24/01/09 09:59:45 INFO DAGScheduler: Job 0 finished: collect at /home/hadoop/spark/word_count.py:15, took 4.649343 s
Today: 1
we: 3
are: 2
learning: 1
Apache: 1
Spark: 1
: 3
We: 1
going: 1
to: 2
creat: 1
python: 1
scripts: 1
shall: 2
also: 1
see: 2
scala: 1
which: 1
is: 1
there: 1
by: 1
default: 1
Later: 1
on: 1
how: 1
use: 1
JDBC: 1
connect: 1
```



```
connect: 1
and: 1
user: 1
API's: 1
24/01/09 09:59:45 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4041
24/01/09 09:59:45 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/01/09 09:59:46 INFO MemoryStore: MemoryStore cleared
24/01/09 09:59:46 INFO BlockManager: BlockManager stopped
24/01/09 09:59:46 INFO BlockManagerMaster: BlockManagerMaster stopped
24/01/09 09:59:46 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/01/09 09:59:46 INFO SparkContext: Successfully stopped SparkContext
24/01/09 09:59:47 INFO ShutdownHookManager: Shutdown hook called
24/01/09 09:59:47 INFO ShutdownHookManager: Deleting directory /tmp/spark-91544640-26e0-46c3-833a-f915f67dd0bf
24/01/09 09:59:47 INFO ShutdownHookManager: Deleting directory /tmp/spark-da4a9b14-17e8-4b9b-bd71-5361379ad089
24/01/09 09:59:47 INFO ShutdownHookManager: Deleting directory /tmp/spark-da4a9b14-17e8-4b9b-bd71-5361379ad089/pyspark
-20cf107f-e613-45b7-978b-90e3a05acc5e
```

Using Spark SQL
$ vim input.csv

```
slno,items,price,type
1,Books,25.0,stationery
2,Pens,10.0,stationery
3,sugar,45.0,House Item
4,Furniture,10000.0,Home interiors
```

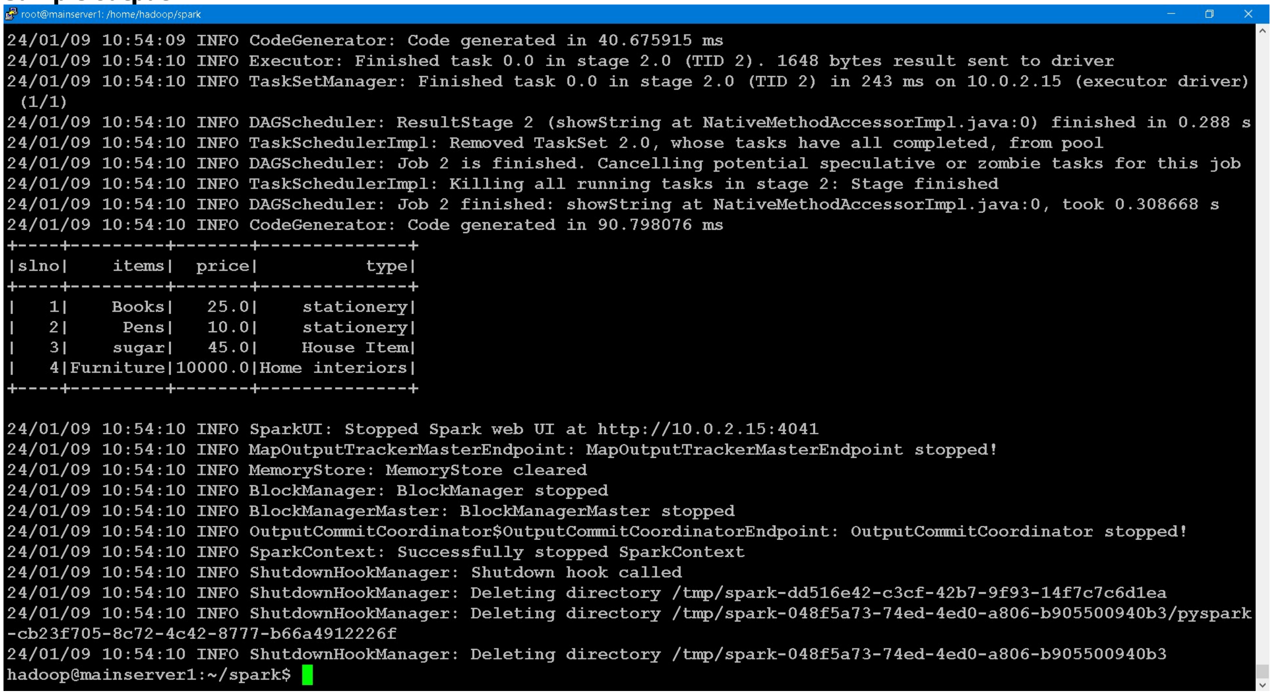
<save and exit>

$ vim spark_sql_sample.py

```
from pyspark.sql import SparkSession

# Create a Spark session
spark = SparkSession.builder.appName("SparkSQLExample").getOrCreate()

# Read a CSV file into a DataFrame
df = spark.read.csv("/home/hadoop/spark/input.csv", header=True, inferSchema=True)

# Perform SQL queries
df.createOrReplaceTempView("my_sparksql_1")
result = spark.sql("SELECT * FROM my_sparksql_1")

# Show the result
result.show()

# Stop the Spark session
spark.stop()
```

<Save and exit>
$ spark-submit spark_sql_sample.py

Sample output



```
24/01/09 10:54:09 INFO CodeGenerator: Code generated in 40.675915 ms
24/01/09 10:54:10 INFO Executor: Finished task 0.0 in stage 2.0 (TID 2). 1648 bytes result sent to driver
24/01/09 10:54:10 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 2) in 243 ms on 10.0.2.15 (executor driver)
 (1/1)
24/01/09 10:54:10 INFO DAGScheduler: ResultStage 2 (showString at NativeMethodAccessorImpl.java:0) finished in 0.288 s
24/01/09 10:54:10 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
24/01/09 10:54:10 INFO DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
24/01/09 10:54:10 INFO TaskSchedulerImpl: Killing all running tasks in stage 2: Stage finished
24/01/09 10:54:10 INFO DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took 0.308668 s
24/01/09 10:54:10 INFO CodeGenerator: Code generated in 90.798076 ms
+----+---------+-------+--------------+
|slno|    items|  price|          type|
+----+---------+-------+--------------+
|   1|    Books|   25.0|    stationery|
|   2|     Pens|   10.0|    stationery|
|   3|    sugar|   45.0|    House Item|
|   4|Furniture|10000.0|Home interiors|
+----+---------+-------+--------------+

24/01/09 10:54:10 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4041
24/01/09 10:54:10 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/01/09 10:54:10 INFO MemoryStore: MemoryStore cleared
24/01/09 10:54:10 INFO BlockManager: BlockManager stopped
24/01/09 10:54:10 INFO BlockManagerMaster: BlockManagerMaster stopped
24/01/09 10:54:10 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/01/09 10:54:10 INFO SparkContext: Successfully stopped SparkContext
24/01/09 10:54:10 INFO ShutdownHookManager: Shutdown hook called
24/01/09 10:54:10 INFO ShutdownHookManager: Deleting directory /tmp/spark-dd516e42-c3cf-42b7-9f93-14f7c7c6d1ea
24/01/09 10:54:10 INFO ShutdownHookManager: Deleting directory /tmp/spark-048f5a73-74ed-4ed0-a806-b905500940b3/pyspark
-cb23f705-8c72-4c42-8777-b66a4912226f
24/01/09 10:54:10 INFO ShutdownHookManager: Deleting directory /tmp/spark-048f5a73-74ed-4ed0-a806-b905500940b3
hadoop@mainserver1:~/spark$
```