# BigData and Machine Learning with Hadoop and Spark Frameworks

Jean-Marc GRATIEN[1]

[1] Department of Computer Science
IFP New Energy

January 20th 2020 / Master Data-AI

# Outline I

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX

# Outline II

4 TP
- TP0 : Hadoop Installation
- TP1 : Hadoop World count
- TP2 : Hadoop DataBase request
- TP3 : Start with Spark
- TP4 : Spark ML, Data processing
- TP5 : Spark ML, Machine learning
- TP6 : Spark ML, Image processing

# Objectifs
## Objectifs

- General Overview on Hadoop and Spark
- Introduce to Hadoop
- Introduction to Spark Framework

# Audience and Prerequisites

- Audience : computer science and data scientist students
- Prerequisites :
  - sequential programming in java and python
  - elementary of machine learning, data analytics
  - image processing
- Material(Slide+TPs) available at :
  `https://drive.google.com/open?id=1HRx6qPRVYckY8H7KMdAcADWpI9iB-bl9`

# Motivation
## Introduction to Bigdata

BigData

- What is Bigdata?
- What are the BigData issues?

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Hadoop Framework

## Introduction to Hadoop

- Hadoop definition
  - Java opensource software framework
  - Data storage management
  - Parallel data analysis
  - part of Apache project supported by the Apache Software Foundation

# Hadoop Framework

## Introduction to Hadoop

- Hadoop History
  - 1990 - 2000 : World Wide Web
  - Yahoo, AltaVisa,. . . : first search engines
  - Nutch open source project created by Doug Cutting and Mike Cafarella
  - 2006 : Nutch project is split:
    the distributed storage and computing framework -> Hadoop
  - 2008 : Hadoop 1.0 (Open Source Project proposed by Yahoo)
  - 2012 : Hadoop 2.0 release
  - 2017 : Hadoop 3.0

# Hadoop Framework

## Introduction to Hadoop

Why Hadoop?

- BigData issues :
  - increasing amount of data amount
  - distributed storage facilities
  - parallel data processing management
  - fault tolerance management

# Outline

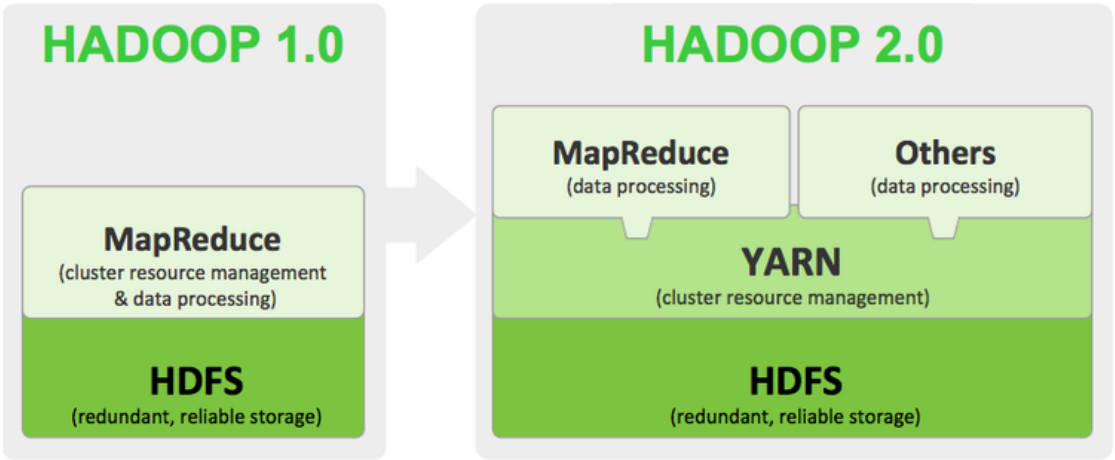
1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Hadoop Framework

## Hadoop Framework Architecture

the basic Hadoop Framework based of 4 main modules :

- HDFS : Hadoop Distributed File System
- YARN : Yest Another Ressource Negotiator
- Map : parallel data processing
- Reduce : collecting data and producing results



# Hadoop Framework

## Hadoop Ecosystem

Hadoop ecosystem :

- Ambari : Hadoop component and services web interface management
- Cassandra : Distributed Data Base system
- Flume : Data Stream management layer
- HBase : NoSql distributed Data Base
- HCatalog : data storage management
- Hive : data storage with a SQL API
- Oozie : task framework
- Pig : HDFS data processing framework
- Solr : data indexing framework
- Sqoop : SQL DB and Hadoop data transfer framework
- Zookeeper : distributed data processing management

# Hadoop Framework
## Hadoop distributions

Hadoop Distributions :


- Hortonworks
- Cloudera
- MAPR

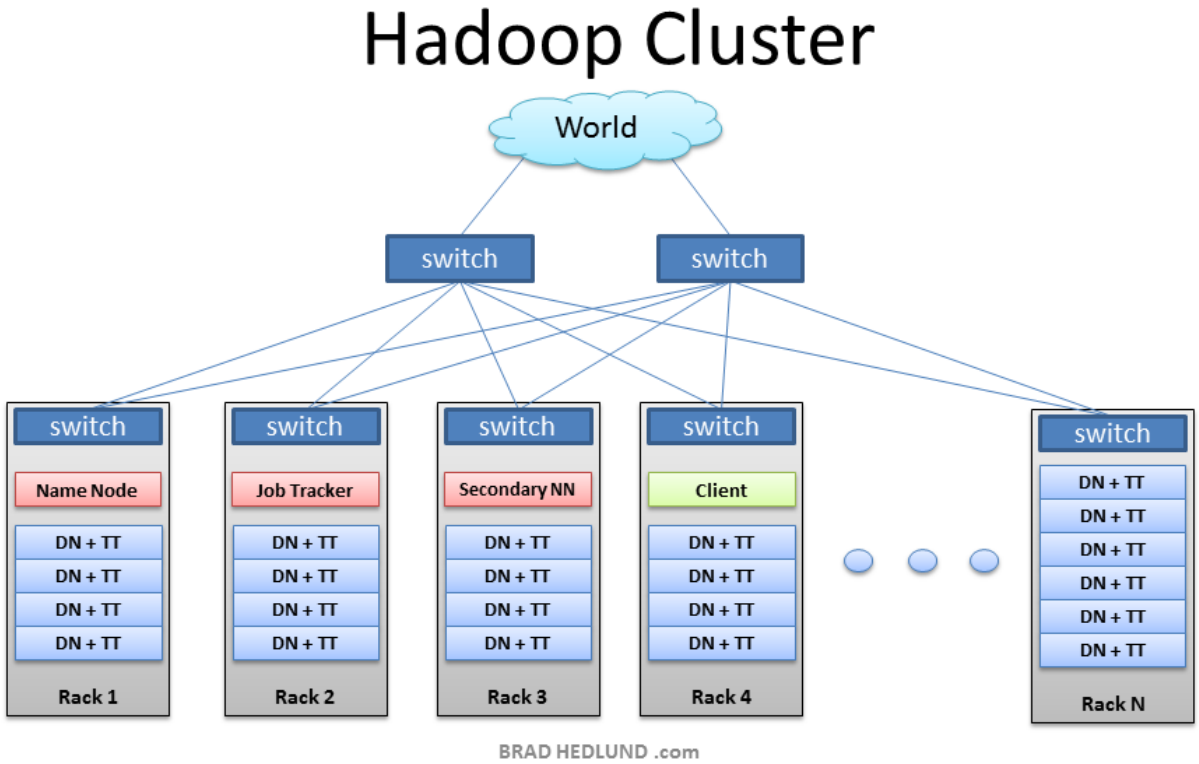


# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Hadoop Framework

## Hadoop Cluster



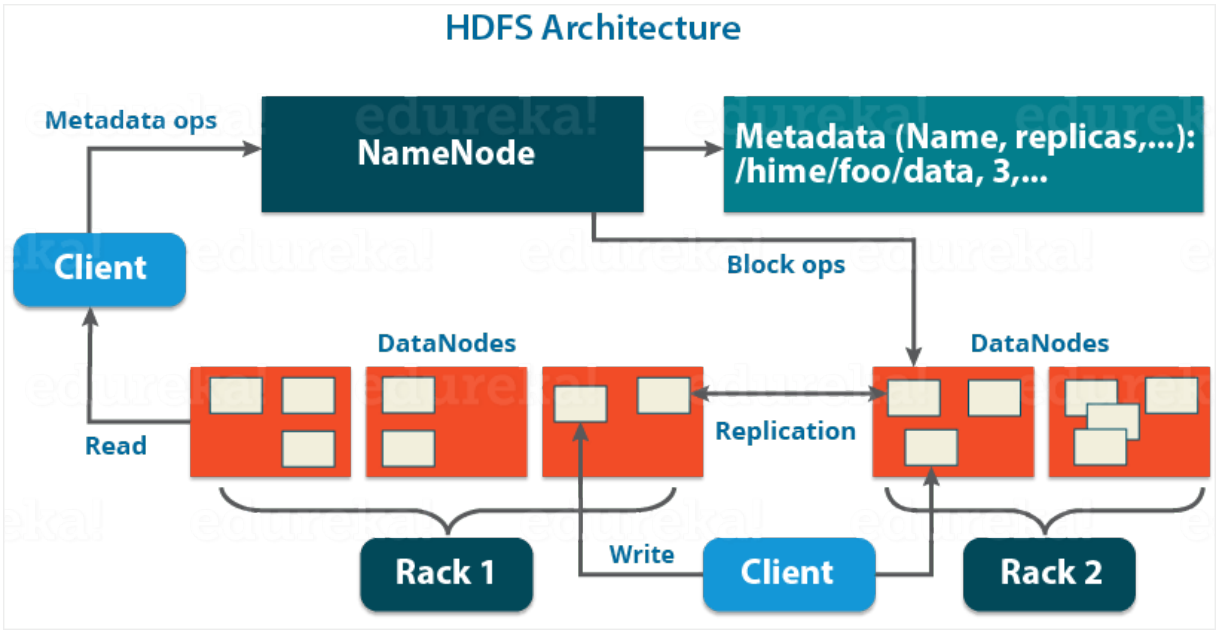

Purpose : Scalability, Fault tolerance and Reliality management

# Hadoop Framework

## Hadoop Cluster

Concepts

- NameNode
- DataNode
- Replication
- Block, Blocksize




HDFS : Hadoop Distributed Filesystem

# Hadoop Framework: HDFS

## HDFS commands

HDFS commands :

### Starting HDFS

```
1 # Format nodes
2 > hadoop namenode -format
3
4 # Starting HDFS services
5 > start-dfs.sh
```

### Shutting down HDFS

```
1 # Stopping HDFS services
2 > stop-dfs.sh
```

# Hadoop Framework: HDFS

## HDFS commands

HDFS commands :

### Inserting Data into HDFS

```
1  # Step 1 : Create input directory
2  > $HADOOP_HOME/bin/hadoop fs -mkdir /usr/input
3
4  # Step 2 : copy data from local filesystem to hdfs
       filesystem
5  > $HADOOP_HOME/bin/hadoop fs -put /home/file.txt /
       user/input
6
7  # Step 3 : check results with ls cmd
8  > $HADOOP_HOME/bin/hadoop fs -ls /usr/input
```

# Hadoop Framework: HDFS

## HDFS commands

HDFS commands :

### Retreiving Data from HDFS

```
1  # Step 1 : view data
2  > $HADOOP_HOME/bin/hadoop fs -cat /user/outputfile
3
4  # Step 2 : get data from hdfs filesystem to local
       filesystem
5  > $HADOOP_HOME/bin/hadoop fs -get /user/output/ /
       home/hadoop_out
6
7  # Step 3 : check results with ls cmd
8  > $HADOOP_HOME/bin/hadoop fs -mkdir /usr/input
```

# Hadoop Framework: HDFS

## HDFS commands list

| Commande name | Description |
|---|---|
| fs -help <cmd-name> | return cmd usage |
| fs -ls <path> | list <path> directory contents |
| fs -lsr <path> | ls ,recursively with sub dirs |
| fs -du <path> | show disk usage in bytes |
| fs -dus <path> | show disk usage in bytes and summary |
| fs -test [ezd] <path> | return 1 if path exists;<br>has 0 length; or is a directory,<br>otherwize 0 |
| fs -cat <filename> | |
| fs -tail [-f] <filename> | |

# Hadoop Framework: HDFS

## HDFS commands list

| Commande name | Description |
|---|---|
| fs -mv <src><dest> | move file or directory within HDFS |
| fs -cp <src> <dest> | copy file or directory within HDFS |
| fs -rm <path> | remove file or directory within HDFS |
| fs -rmr <path> | rm recursively |
| fs -put <localSrc> <dest> | copy files or dirs from local FS to HDFS |

# Hadoop Framework: HDFS

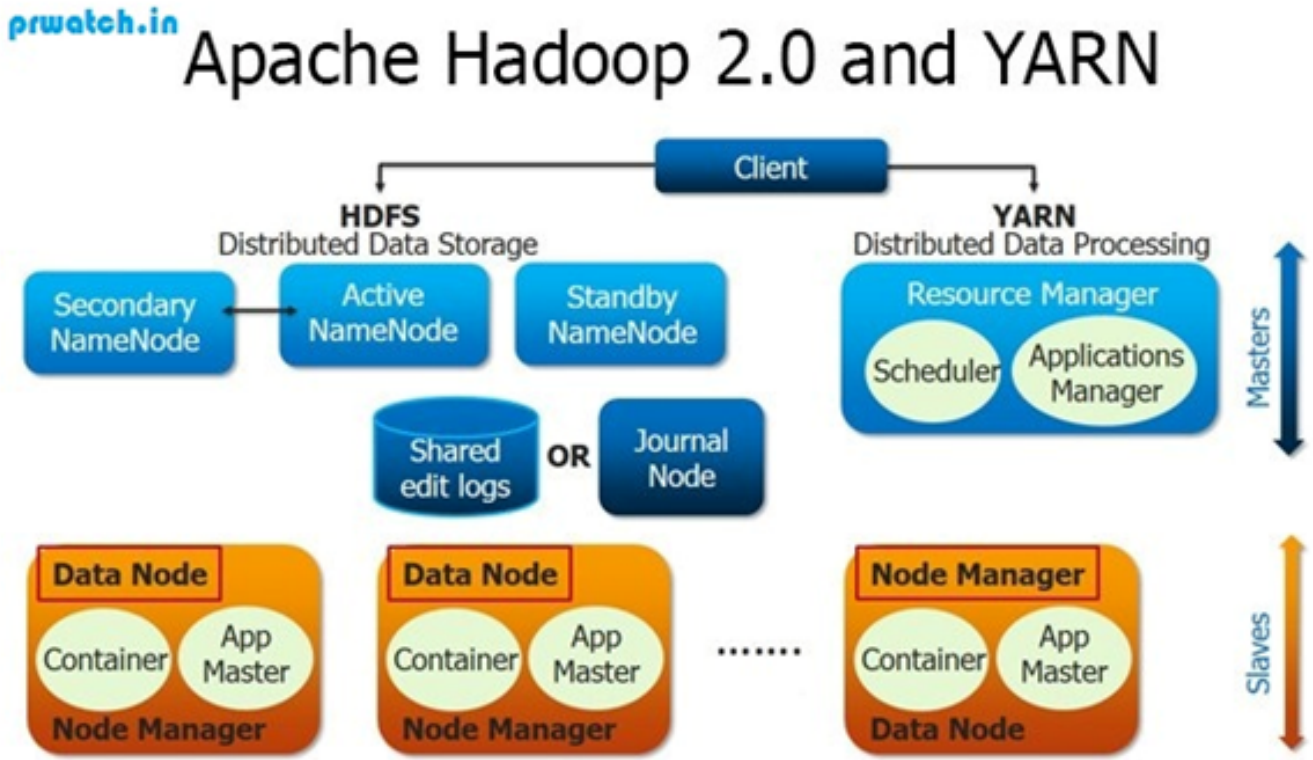
## HDFS commands list

| Commande name | Description |
|---|---|
| fs -copyFromLocal <localSrc> <dest> | identical to put |
| fs -moveFromLocal <localSrc> <dest> | move file or dirs from local FS to HDFS |
| fs -get [-crc] <src> <localDest> | copy file or dirs from HDFS to local FS |
| fs -getmerge [-crc] <src> <localDest> | copy all files from HDFS and merge to a single file in FS |
| fs -copyToLocal <localSrc> <dest> | copy file or dirs from HDFS to local FS |
| fs -moveToLocal <localSrc> <dest> | move file or dirs from HDFS to local FS |
| fs -mkdir <path> | create directory in HDFS |

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Hadoop Framework
## YARN



# Hadoop Framework
## YARN

### Starting YARN

```
1 # Starting YARN services
2 > start-yarn.sh
```

### Shutting down YARN

```
1 # Stopping YARN services
2 > stop-yarn.sh
```

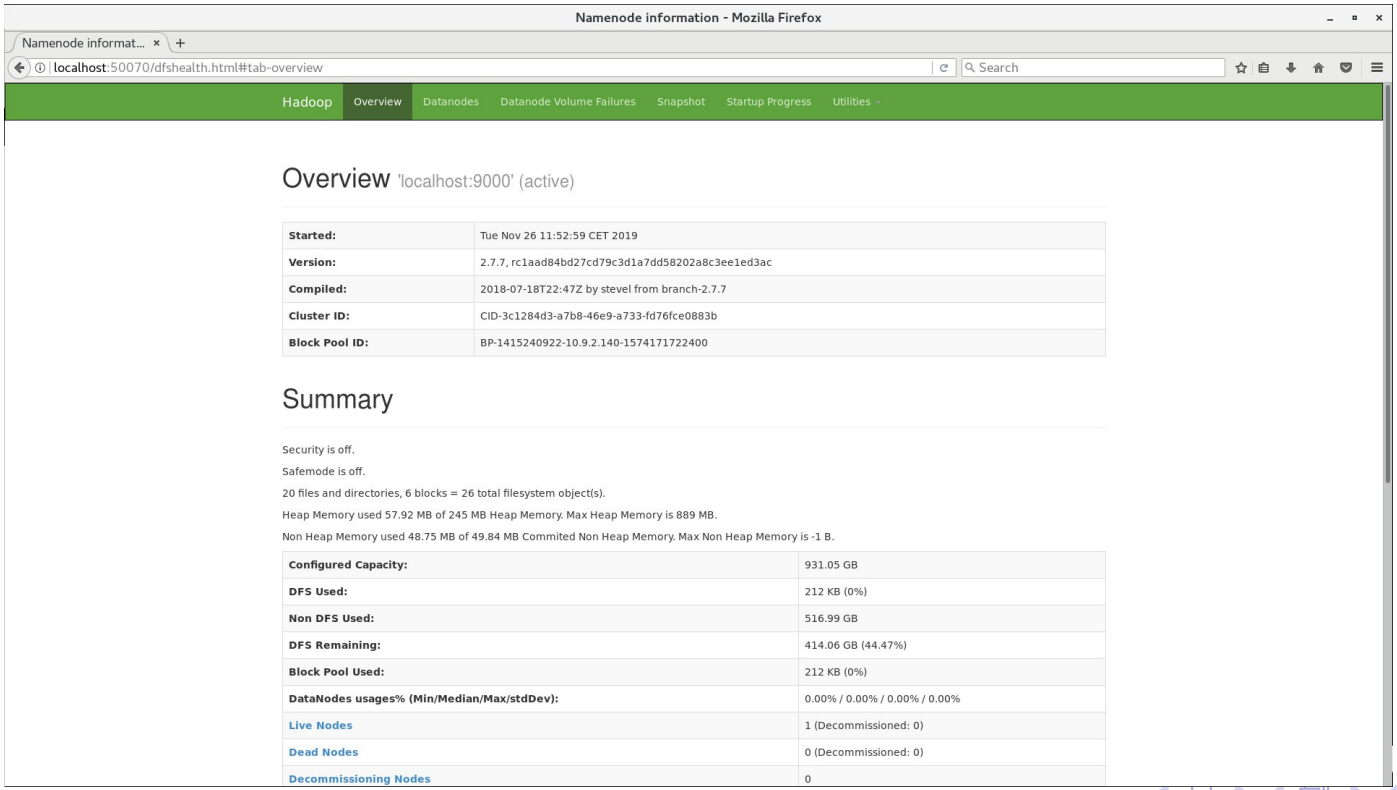
# Hadoop Web tools

## Hadoop Web Tools

Web tools on : `http://<hostname>:<port>`

<hostname>:<port> (default localhost:50070) are defined in hdfs-site.xml

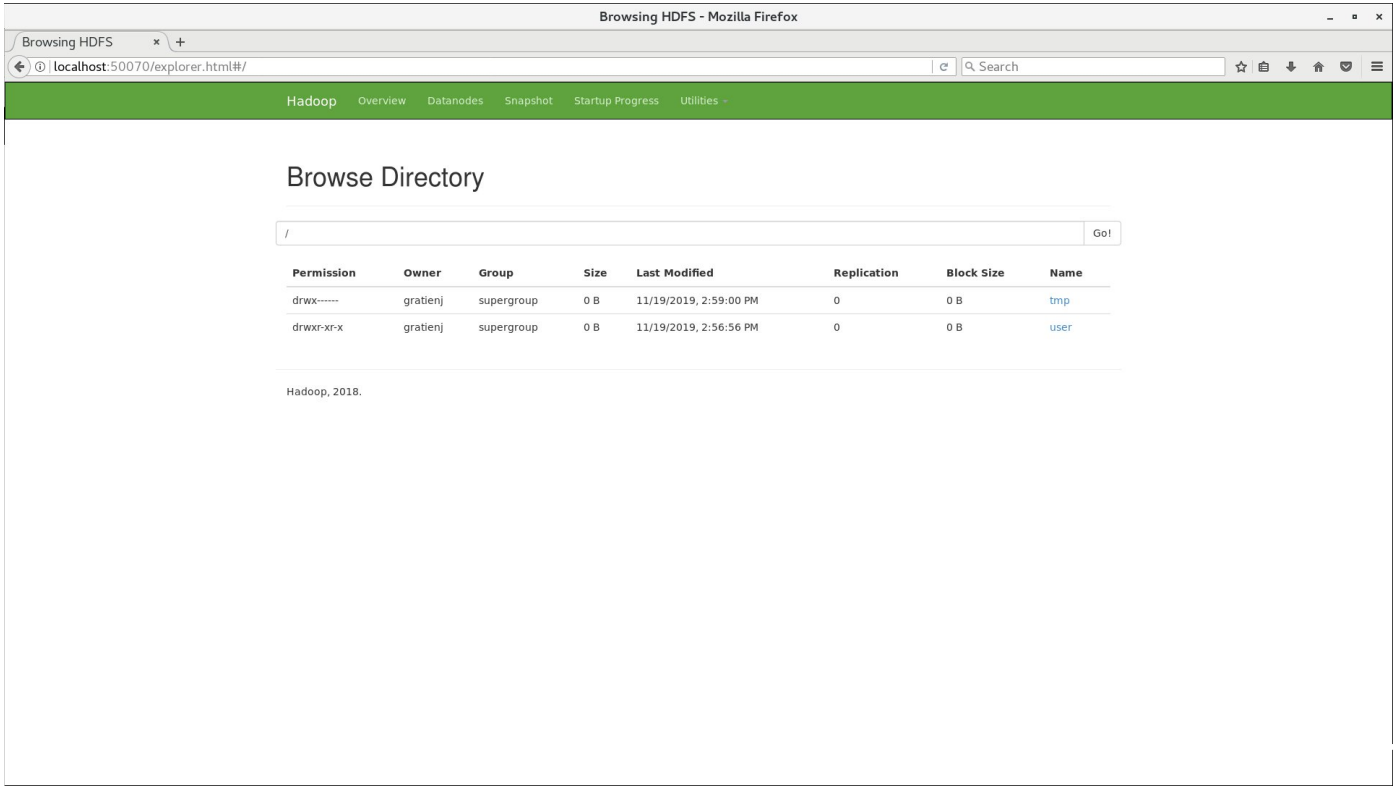
Overview web page:



# Hadoop Web tools

## Hadoop Web Tools

File browser web page:

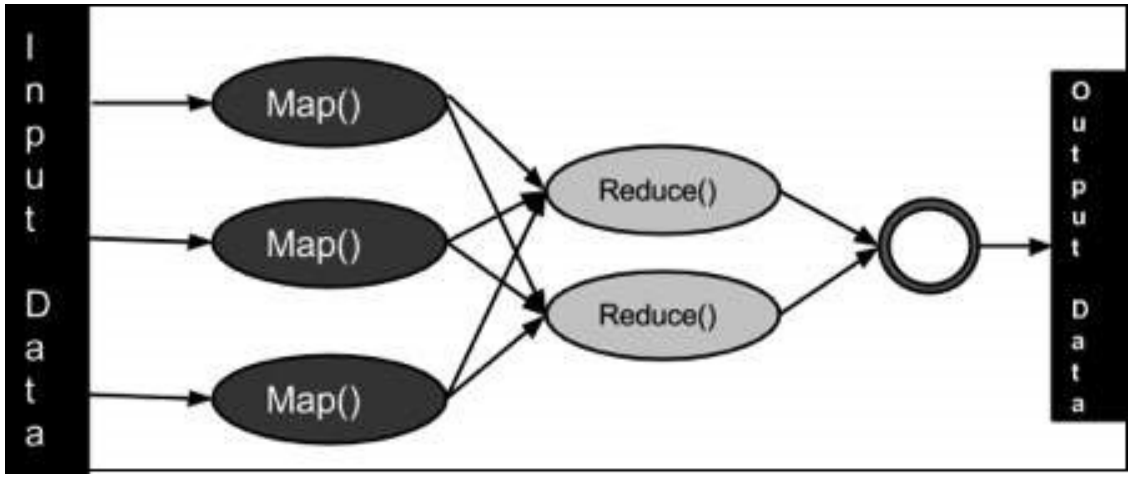


# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Hadoop Framework

## MapReduce Framework




MapReduce Framework

# Hadoop Framework

## MapReduce Framework

MapReduce Algorithm :

- Programming model ;
- Two stages:
  - Map stage :
    - Mapper jobs;
    - data are processed in parallel by mapper jobs;
  - Reduce Stage :
    - Reducer jobs;
    - mapper output data are processed Reducer jobs;
    - Reducer jobs produce new set of output stored in HDFS.

# Hadoop Framework

## MapReduce Framework

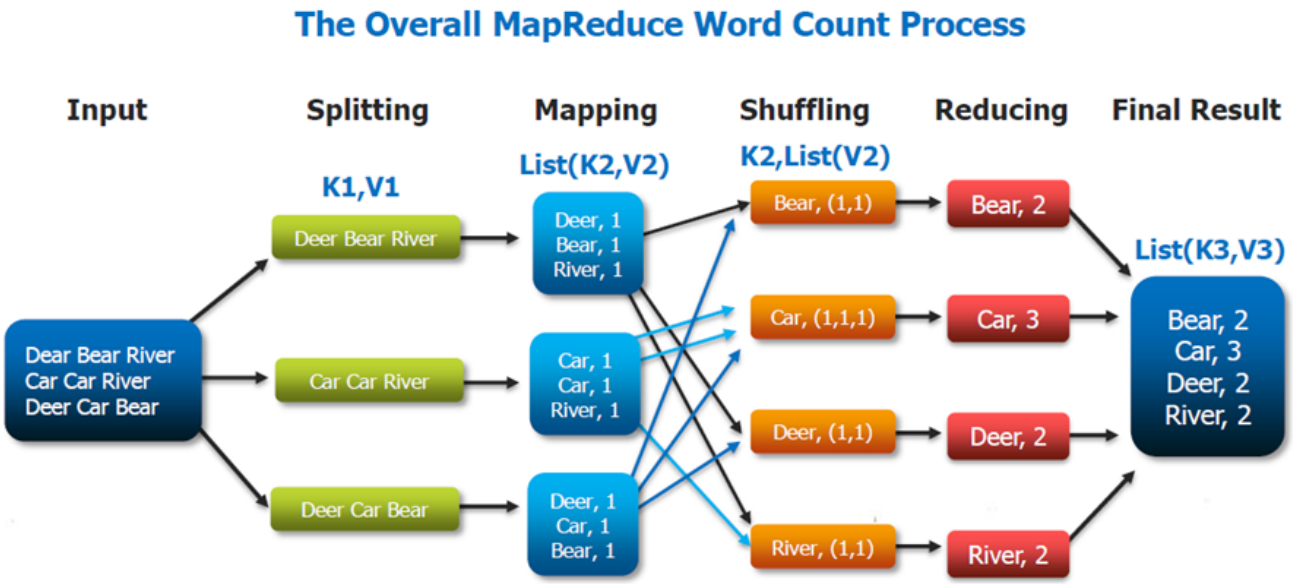
Input and Output :

- Input : <key,value> pairs
  - key and values classes must implement Writable Interface;
  - key class have to implemente WritableComparable Interface;
- Job : (Input)$\rightarrow$ map $\rightarrow$ <k2,v2> $\rightarrow$ reduce $\rightarrow$ <k3,v3>(Output)

| | Input | Output |
|---|---|---|
| Map | <k1,v1> | list(<k2,v2>) |
| Reduce | <k2,v2> | list(<k3,v3>) |

# Hadoop Framework

## MapReduce Framework



# Hadoop Framework

## Java exemple

### WordCount Java class

```
1  import org.apache.hadoop.*;
2  public class WordCount  {
3    public static class  TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
4      public void map(Object key, Text value, Context context)
5                      throws IOException, InterruptedException {
6          ... ;
7      }
8    }
9    public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable>  {
10     public void reduce(Text key, Iterable<IntWritable> values,Context context)
11                       throws IOException, InterruptedException  {
12           ... ;
13     }
14   }
15   public static void main(String[] args) throws Exception {
16   ...;
17   }
18 }
```

# Hadoop Framework
## Java exemple

### Word Count Mapper Java class

```
1  public class WordCount
2  {
3    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>
4    {
5      public void map(Object key, Text value, Context context)
6                      throws IOException, InterruptedException
7      {
8        StringTokenizer itr = new StringTokenizer(value.toString());
9        while (itr.hasMoreTokens()) {
10         word.set(itr.nextToken());
11         context.write(word, one);
12       }
13     }
14   }
15 }
```

# Hadoop Framework
## Java exemple

### WordCount Reducer Java class

```
1  public class WordCount
2  {
3     public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable>
4     {
5          private IntWritable result = new IntWritable();
6          void reduce(Text key, Iterable<IntWritable> values,Context context)
7                    throws IOException, InterruptedException
8          {
9             int sum = 0;
10            for (IntWritable val : values) {
11               sum += val.get();
12            }
13            result.set(sum);
14            context.write(key, result);
15         }
16    }
17 }
```

# Hadoop Framework

## Java exemple

### Main test function

```
1  public class WordCount
2  {
3    public static void main(String[] args) throws Exception
4    {
5      Configuration conf = new Configuration();
6      Job job = Job.getInstance(conf, "word count");
7      job.setJarByClass(WordCount.class);
8      job.setMapperClass(TokenizerMapper.class);
9      job.setCombinerClass(IntSumReducer.class);
10     job.setReducerClass(IntSumReducer.class);
11     job.setOutputKeyClass(Text.class);
12     job.setOutputValueClass(IntWritable.class);
13     FileInputFormat.addInputPath(job, new Path(args[0]));
14     FileOutputFormat.setOutputPath(job, new Path(args[1]));
15     System.exit(job.waitForCompletion(true) ? 0 : 1);
16   }
17 }
```

# Hadoop Framework

## Java exemple

Suppose we have two test files file01 and file02 in current directory

### Prepare Test Data

```
1  // Two test file file01 file02 in current directory
2  $ ls
3     file01
4     file02
5  $ hdfs dfs -put file01 /user/gratienj/input
6  $ hdfs dfs -cat /user/gratienj/input/file01
7     Hello World Bye World
8  $ hdfs dfs -put file02 /user/gratienj/input
9  $ hdfs dfs -cat /user/gratienj/input/file02
10   Hello Hadoop Goodbye Hadoop
```

# Hadoop Framework

## Java exemple

Suppose the Java Project is compiled and generates the jar file BigDataTP1.jar

### Run application

```
$ hadoop jar BigDataTP1.jar hadoop.WordCount /user/gratienj/input /user/gratienj/output
```

### Check results

```
$ hdfs dfs -cat /user/gratienj/output/part-r-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

# Hadoop Framework
## Python example

Python example with Hadoop Streaming

### Mapper python script

```
1  #!/usr/bin/env python
2  """mapper.py"""
3  import sys
4  # input comes from STDIN (standard input)
5  for line in sys.stdin:
6      line = line.strip()
7      words = line.split()
8      for word in words:
9          print('%s\t%s' % (word, 1))
```

# Hadoop Framework
## Python example

### Reducer python script Part 1

```
1 from operator import itemgetter
2 import sys
3 current_word = None
4 current_count = 0
5 word = None
```

# Hadoop Framework

## Python example

### Reducer python script Part 2

```
1  for line in sys.stdin: # input comes from STDIN
2      line = line.strip()
3      word, count = line.split('\t', 1)
4      try:
5          count = int(count)
6      except ValueError:
7          continue
8      if current_word == word:
9          current_count += count
10     else:
11         if current_word:
12             print('%s\t%s' % (current_word, current_count)) # write result to STDOUT
13         current_count = count
14         current_word = word
15 if current_word == word: # do not forget to output the last word if needed!
16     print('%s\t%s' % (current_word, current_count))
```

# Hadoop Framework
## Python example

### Python example with Hadoop Streaming : Part 1

#### Copy test files on HDFS

```
1 $ hdfs dfs -copyFromLocal /home/gratienj/test/books /user/gratienj/input/books
```

#### Run application

```
1 $ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.7.7.jar \
2  -file /home/hduser/mapper.py    -mapper /home/hduser/mapper.py \
3  -file /home/hduser/reducer.py   -reducer /home/hduser/reducer.py \
4  -input /user/gratienj/input/books/* -output /user/gratienj/books-output
```

# Hadoop Framework
## Python example

Python example with Hadoop Streaming : Part 2

### Check results

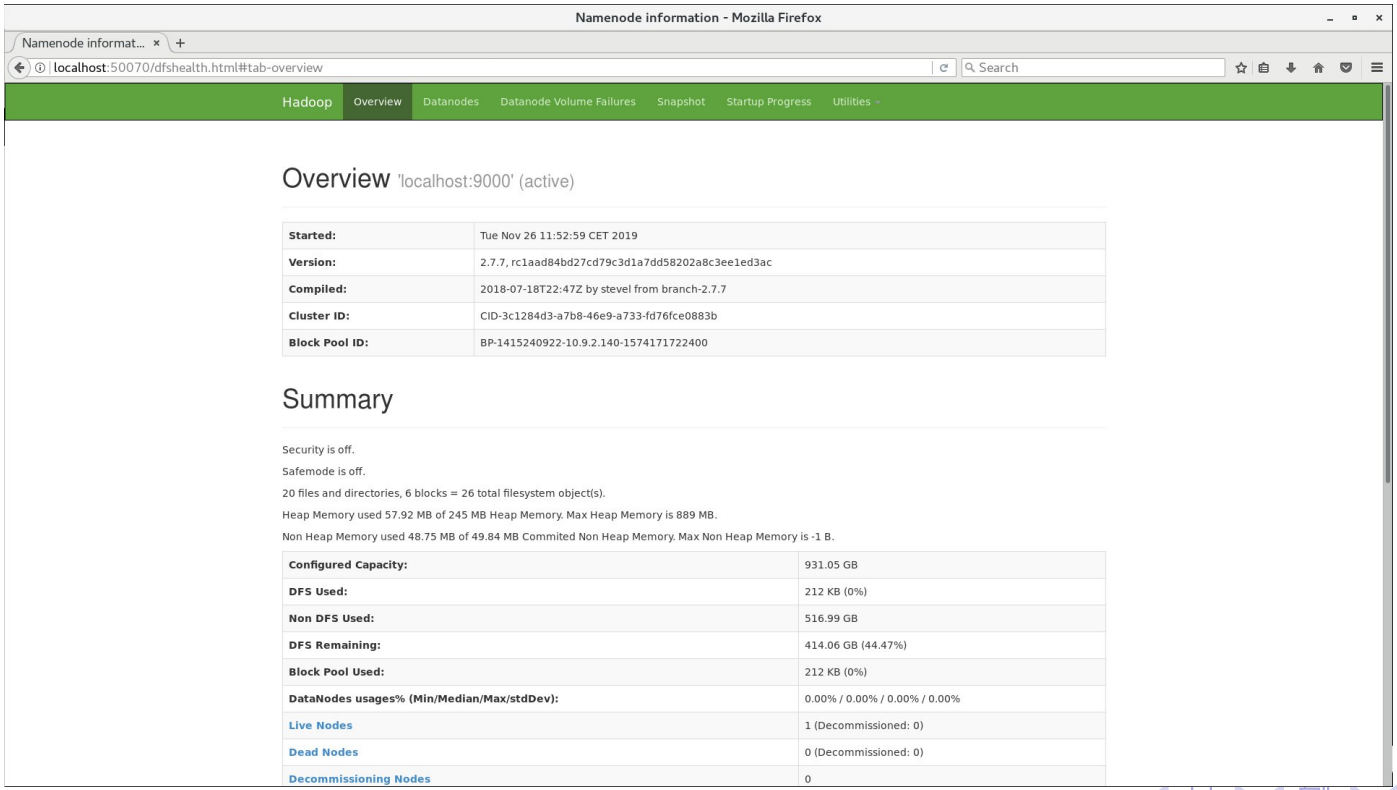
```
$ hdfs dfs -ls /user/gratienj/books-ouput
Found 1 items
/user/gratienj/books-output/part-00000
$ hdfs dfs -cat /user/gratienj/books-output/part-00000
```

# Hadoop Web tools

## Ambari Server Tools

Ambari Server : tools to manage and monitor applications for Apache Hadoop

Web page : `http://<ambari-server-hostname>:8080`



# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Spark Framework
## Introduction to Spark

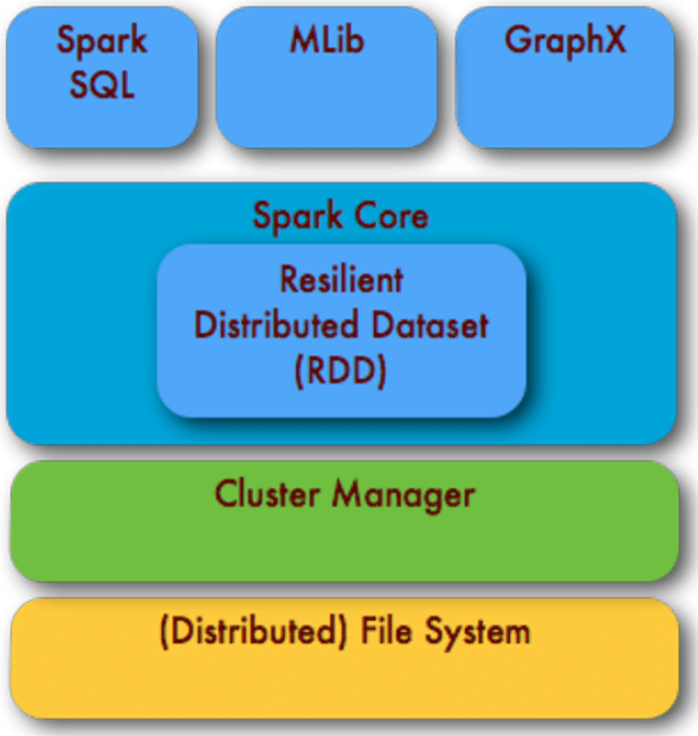
- Spark : Big Data framework for data processing
- History
  - 2009 : AMPLab, UC Berkeley University
  - 2010 : Open source as an Apache project
- Complete and Unified framework
  - Hadoop (MapReduce)
  - Storm (Streaming)
  - Languages : Java, Scala, Python
  - SQL

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

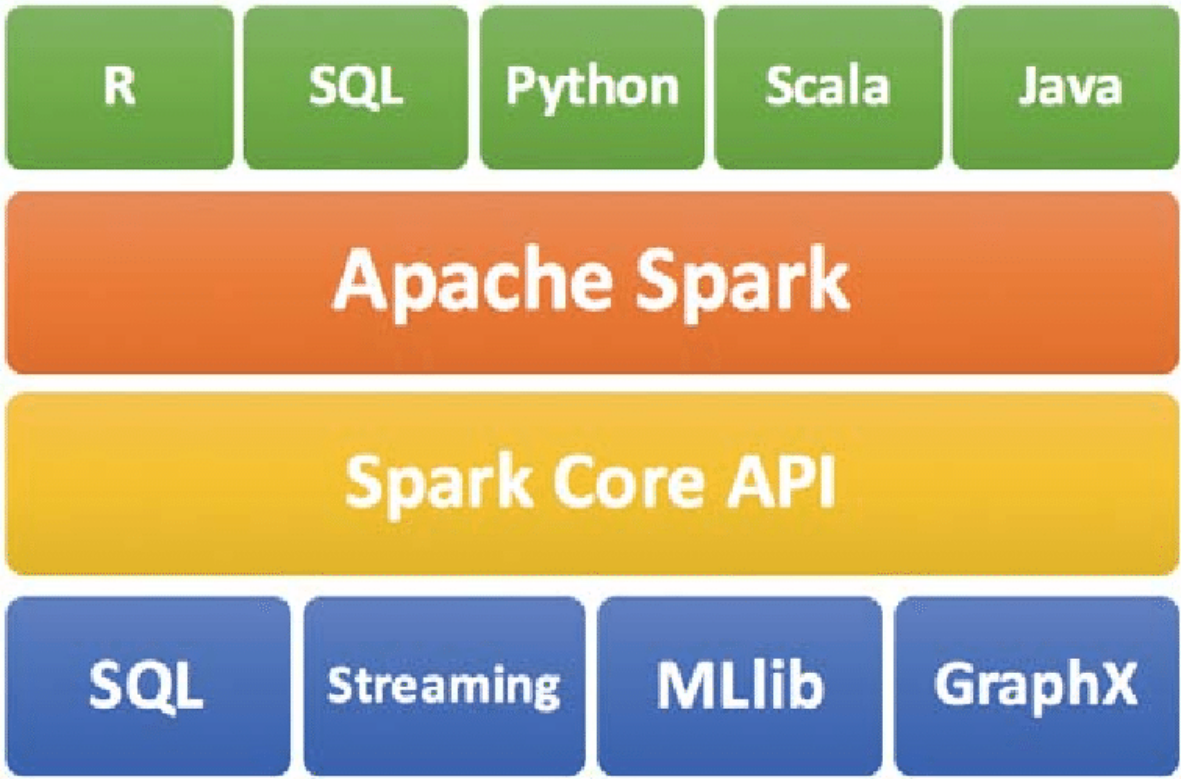
# Spark Framework

## Apache Spark Architecture



# Spark Framework

## Apache Spark Ecosystem



# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Spark Framework

## Spark Core : Spark configuration

Spark Cluster Configurations :

- Local mode
- Cluster mode
- Client mode

Spark parallel concepts:

- multiple executors (private JVM)
- multiple cores per executor

# Spark Framework

## Spark Core : Spark configuration

### Configuring a SparkContext

```
1  import pyspark
2  from pyspark import SparkConf
3  sc_conf = SparkConf()
4  sc_conf.setAppName(app_name)
5  sc_conf.setMaster('local[*]')
6  sc_conf.set('spark.executor.memory', '4g')
7  sc_conf.set('spark.executor.cores', nb_cores)
8  sc_conf.set('spark.driver.memory', '16G')
9  sc_conf.set('spark.cores.max', '32')
10 sc_conf.set('spark.driver.maxResultSize', '10G')
11 sc_conf.set('spark.logConf', True)
```

# Spark Framework

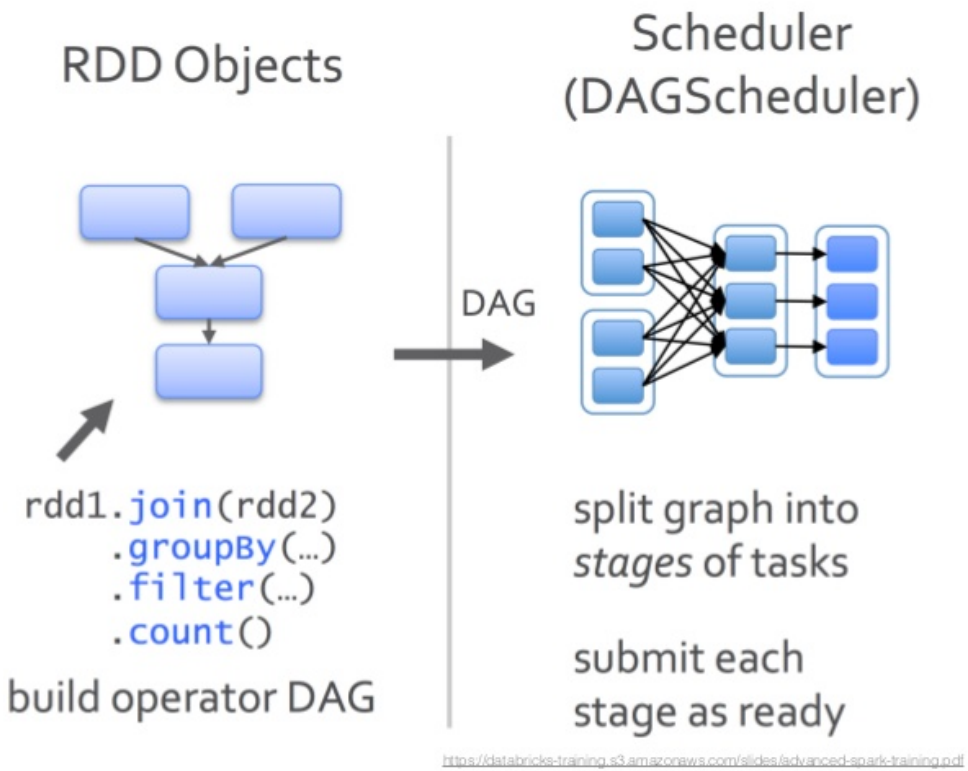
## Spark Core : Spark context

### Create a SparkContext

```
1 import pyspark
2 from pyspark import SparkContext
3 sc =SparkContext()
```

# Spark Framework

## Spark Core : Data concepts



# Spark Framework

## Spark Core : Data concepts

Spark Data concepts :

- **RDD** : Resilient Distributed Data, list of <key,value>
- **Transformations** : apply lambda to creating new RDDs
- **DAG** : pipeline of transformation
- **Actions** : operations on the RDD producing results
- **Scheduler** : perform actions on DAG
- **Stage** : parallel operations
- **Pipeline** : sequence of stages

# Spark Framework

## Spark Core : RDD

### Create a Spark RDD

```
1 import pyspark
2 from pyspark import SparkContext
3 sc =SparkContext()
4
5 nums= sc.parallelize([1,2,3,4])
6
7 nums.take(1)
```

### Output

```
1 [1]
```

# Spark Framework

## Spark Core : RDD Transformationq and Actions

### Spark RDD transformations and Actions

```
1 sc =SparkContext()
2 nums= sc.parallelize([1,2,3,4])
3 squared = nums.map(lambda x: x*x).collect()
4 for num in squared:
5     print('%i ' % (num))
```

### Output

```
1 1
2 4
3 9
4 16
```

# Spark Framework

## Spark Core : RDD Transformations and Actions

Transformation :

- apply lambda function to RDD
- create a new RDD
- lazy evaluation
- create a DAG

# Spark Framework

## Spark Core : RDD Transformations and Actions

Examples :

| Commande name | Description |
|---|---|
| map() | apply to each RDD line |
| flatMap() | apply to all RDD elements |
| mapPartition | apply per partition |
| filter() | apply to a selection of lines |
| groupBy() | create new set of (key,value) |
| groupByKey() | |
| reduceByKey() | |
| sample() | selection of lines |
| union() | fusion of two RDDs |
| join() | union without duplicate keys |

# Spark Framework

## Spark Core : RDD Transformations and Actions

Actions :

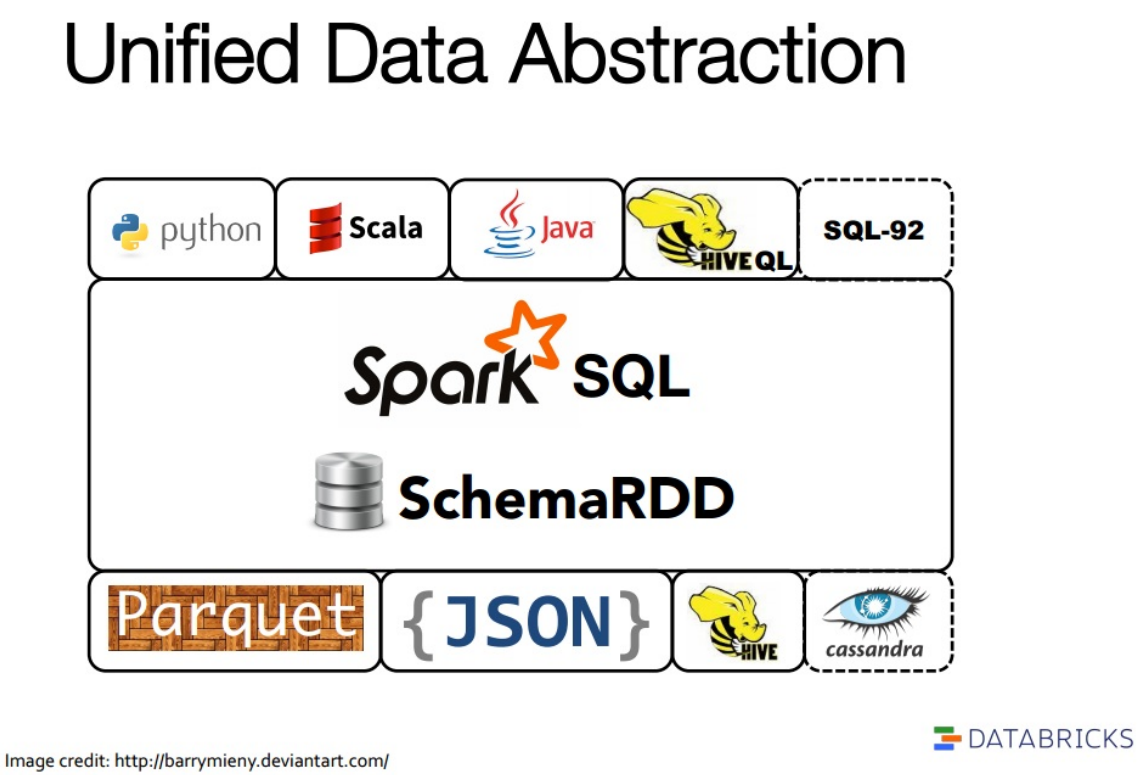
- get results on a pipeline of transformations
- perform all the transformation
- real evaluation

Examples :

| Commande name | Description |
|---|---|
| getNumPartition()<br>reduce()<br><br><br>collect() | <br><br>apply lambda<br>to all elements<br>create a collection |
| count()<br>max(), min()<br>sum() | count elements<br>stats |

# Spark Framework

## Spark SQL : DataFrame



# Spark Framework

## Spark Core : RDD

### Create a Spark SQL context

```
1 from pyspark.sql import Row
2 from pyspark.sql import SQLContext
3 sqlContext = SQLContext(sc)
```

### Create a DataFrame

```
1 list_p=[('John',19),('Smith',29),('Adam',35)]
2 rdd = sc.parallelize(list_p)
3 ppl_rdd=rdd.map(lambda x: Row(name=x[0], age=int(x
    [1])))
4 ppl_df_rdd = sqlContext.createDataFrame(ppl_rdd)
```

# Spark Framework

## Spark SQL : DataFrame

### Print DataFrame Schema

```
1 DF_ppl.printSchema()
2 root
3  |-- age: long (nullable = true)
4  |-- name: string (nullable = true)
```

# Spark Framework

## Spark SQL : DataFrame

### Print DataFrame Schema

```
df = sqlContext.read.csv(SparkFiles.get("adult_data.csv"), header=True, inferSchema= True)
df_string.printSchema()
root
 |-- age: string (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: string (nullable = true)
 |-- education: string (nullable = true)
 |-- education_num: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital_gain: string (nullable = true)
 |-- capital_loss: string (nullable = true)
 |-- hours_week: string (nullable = true)
 |-- native_country: string (nullable = true)
 |-- label: string (nullable = true)
```

# Spark Framework

## Spark SQL : DataFrame

### Select columns

```
1  df.select('age','fnlwgt')
       .show(5)
```

### Select columns

```
+---+------+
|age|fnlwgt|
+---+------+
| 39| 77516|
| 50| 83311|
| 38|215646|
| 53|234721|
| 28|338409|
+---+------+
only showing top 5 rows
```

# Spark Framework

## Spark SQL : DataFrame

### Select columns

```
1 df.groupBy("education").
      count().sort("count",
      ascending=True).show()
```

### Select columns

```
1  +------------+-----+
2  |   education|count|
3  +------------+-----+
4  |   Preschool|   51|
5  |     1st-4th|  168|
6  |     5th-6th|  333|
7  |   Doctorate|  413|
8  |        12th|  433|
9  |         9th|  514|
10 | Prof-school|  576|
11 |     7th-8th|  646|
12 |        10th|  933|
13 |  Assoc-acdm| 1067|
14 |        11th| 1175|
15 |   Assoc-voc| 1382|
16 |     Masters| 1723|
17 |   Bachelors| 5355|
18 |Some-college| 7291|
19 |     HS-grad|10501|
20 +------------+-----+
```

# Spark Framework

## Spark SQL : DataFrame

Describe data: describe() functions give a summary of statistics :

- count,
- mean,min,max
- standarddeviation

### Describe

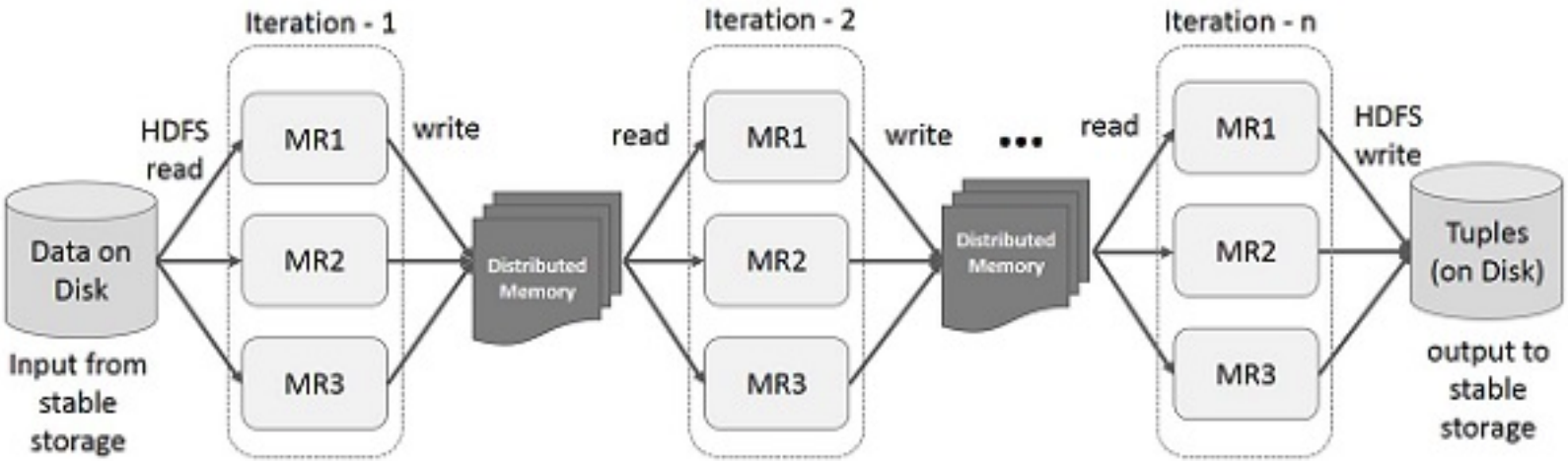
```
1 df.describe('capital_gain
    ').show()
```

### Describe

```
1 +-------+------------------+
2 |summary|      capital_gain|
3 +-------+------------------+
4 |  count|             32561|
5 |   mean|1077.6488437087312|
6 | stddev| 7385.292084840354|
7 |    min|                 0|
8 |    max|             99999|
9 +-------+------------------+
```

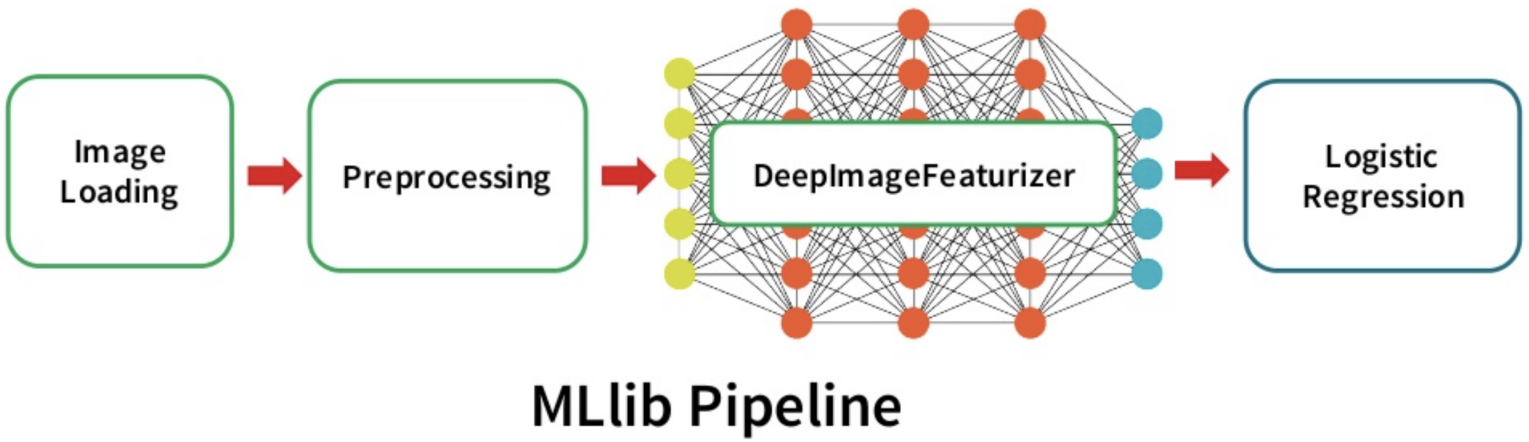
# Spark Framework

## Spark Mlib



# Spark Framework

## Spark Mlib



# Spark Framework
## Spark Mlib

Mlib provides tools for Machine learning

- set of classifier and regression algorithms
- create models from Spark Dataframe
- set of tools to evaluate the predicting models
- concept of pipeline to process Data

# Spark Framework
## Spark Mlib Pipeline

Spark Pipeline : a sequence of stages (Transformer, Estimator)

- String Indexer : convert Categorical Data to numerics
- Standard Scaler on Continuous Values
- VectorAssembler : features must be a dense vector

# Spark Framework

## Spark Mlib : Data processing

### StringIndexer

```
1 from pyspark.ml.feature import StringIndexer
2 df_rdd = ...
3 # Create String Indexer to convert "cat_key" to "
    ind_key"
4 string_indexer = StringIndexer(inputCol="cat_key",
    outputCol="encoded_key")
5 # Create model
6 model = string_indexer.fit(df_rdd)
7 # Transform RDD
8 encoded_df_rdd = model.transform(df_rdd)
```

# Spark Framework

## Spark Mlib : Data processing

### OneHotEncoder

```
1 from pyspark.ml.feature import OneHotEncoder
2 encoder = OneHotEncoder(dropLast=False, inputCol="
    encoded_key", outputCol="vec_key")
3 vec_df_rdd = encoder.transform(encoded_df_rdd)
```

### VectorAssembler

```
1 from pyspark.ml.feature import  VectorAssembler
2 assembler = VectorAssembler(inputCols=[key1,key2,
    key3], outputCol="features")
3 ass_df_rdd = assembler.transform(df_rdd)
```

# Spark Framework

## Spark Mlib : Data processing

### Pipeline

```
1  from pyspark.ml import Pipeline
2  # DEFINE LIST OF STAGES
3  stages = [[label_indexer],[cat_key_indexer,encoder
       ],[assembler]]
4
5  # DEFINE PIPELINE
6  pipeline = Pipeline(stages=stages)
7
8  # APPLY PIPELINE
9  pipelineModel = pipeline.fit(df_rdd)
10 model_df_rdd = pipelineModel.transform(rdd_df)
```

# Spark Framework

## Spark ML : ML pipeline Part 1

### create DataFrame

```
1 rdd.map(lambda x: (x["newlabel"], DenseVector(x["
      features"])))
2 sqlContext.createDataFrame(input_data, ["label", "
      features"])
```

### Split data

```
1 randomSplit([.8,.2],seed=1234)
```

# Spark Framework

## Spark ML : ML pipeline Part 2

### Train model

```
1 lr=LogisticRegression(labelCol="label",featuresCol=
      "features",maxIter=10, regParam=0.3)
2 lr.fit()
```

### Make prediction

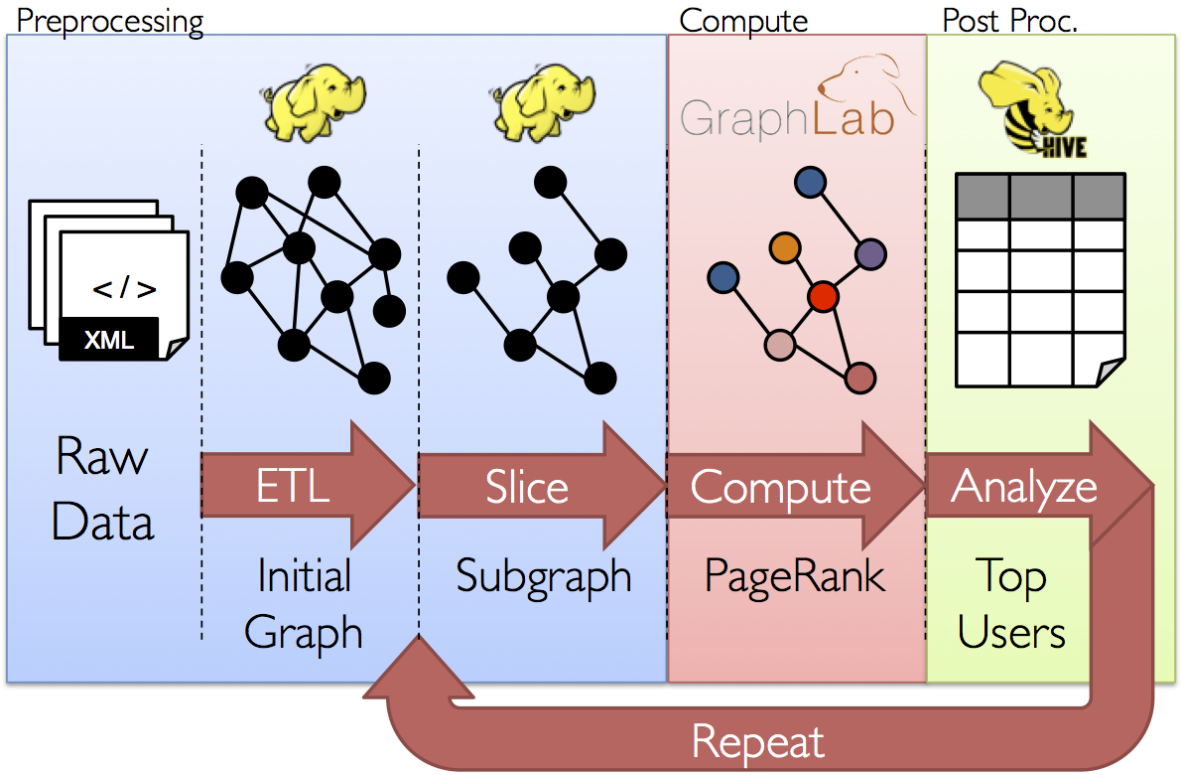
```
1 lr.transform()
```

# Spark Framework

## Spark Streaming

# Spark Framework

## Spark GraphX



# TPs

- TP 0 : Hadoop Installation
- TP 1 : WorldCount
- TP 2 : DataBase request
- TP 3 : Spark Installation
- TP 4 : Spark Compute PI
- TP 5 : Spark Image Processing
- TP 6 : Spark ML

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# TPs

## TP0 : Installation Hadoop

Hadoop Installation: hadoop-2.7.7.tar.gz

### Installation

```
1 > cd /home/hduser
2 > mkdir local ; cd local
3 > wget https://www.apache.org/dist/hadoop/core/
      hadoop-2.7.7/hadoop-2.7.7.tar.gz
4 > tar xvfz hadoop-2.7.7.tar.gz
5 > mv hadoop-2.7.7 hadoop
6 > chown -R hduser:hadoop hadoop
```

# TPs
## TP0 : Installation Hadoop

### Env parameter settings

```
1  # Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
2  export JAVA_HOME=/usr/local/Java/1.8.0-xxx
3
4  # Set Hadoop-related environment variables
5  export HADOOP_HOME=/home/hduser/local/hadoop
6
7  export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
8  export HADOOP_MAPRED_HOME=$HADOOP_HOME
9  export HADOOP_COMMON_HOME=$HADOOP_HOME
10 export HADOOP_HDFS_HOME=$HADOOP_HOME
11 export YARN_HOME=$HADOOP_HOME
12
13 export PATH=${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin:$PATH
```

# TPs
## TP0 : Installation Hadoop

### Configuration files settings

```
1  # CREATE HADOOP TMP DIR
2  > mkdir -p /home/hduser/app/hadoop/tmp
3  > chown hduser:hadoop /home/hduser/app/hadoop/tmp
4  > chmod 750 /home/hduser/app/hadoop/tmp
5
6  # CREATE HDFS WORKINGDIR TO MNG HDFS File System
7  > mkdir -p /home/hduser/var/local/hadoop/hdfs/data
8  > chmod -R 777 /home/hduser/var/local/hadoop/hdfs
```

# TPs

## TP0 : Installation Hadoop

### Configuration files in /home/hduser/local/hadoop/etc/hadoop

#### hadoop-env.sh modification

```
1 JAVA_HOME="true␣java␣JOME␣path"
2 export JAVA_HOME=${JAVA_HOME}
```

#### core-site.xml settings

```
1  <property>
2          <name>hadoop.tmp.dir</name>
3          <value>/home/hduser/app/hadoop/tmp</value>
4          <description>A base for other temporary directories.</description>
5  </property>
6  <property>
7          <name>fs.default.name</name>
8          <value>hdfs://localhost:9000</value>
9          <description>The name of the default file system.</description>
10 </property>
```

# TPs
## TP0 : Installation Hadoop

Configuration files in `/home/hduser/local/hadoop/etc/hadoop`

### hdfs-site.xml settings

```
1  <property>
2      <name>dfs.data.dir</name>
3      <value>/home/hduser/var/local/hadoop/hdfs/data</value>
4      <final>true</final>
5  </property>
6
7  <property>
8          <name>dfs.replication</name>
9          <value>1</value>
10 </property>
```

# TPs
## TP0 : Installation Hadoop

Configuration files in /home/hduser/local/hadoop/etc/hadoop
Copy mapred-site.xml.template mapred-site.xml

### mapred-site.xml settings

```
1 <property>
2         <name>mapred.job.tracker</name>
3         <value>localhost:9001</value>
4 </property>
```

# TPs
## TP0 : Installation Hadoop

Configuration files in /home/hduser/local/hadoop/etc/hadoop

### yarn-site.xml settings

```
1  <configuration>
2          <!-- Site specific YARN configuration properties -->
3          <property>
4                  <name>yarn.nodemanager.aux-services</name>
5                  <value>mapreduce_shuffle</value>
6          </property>
7          <property>
8                  <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
9                  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
10         </property>
11 </configuration>
```

# TPs

## TP0 : Installation Hadoop

### Lauch all services

```
1 > $HADOOP_HOME/sbin/start-hdfs.sh
2 > $HADOOP_HOME/sbin/start-yarn.sh
```

### Check lauched services

```
1 > jps
2
3 26867 DataNode
4 28228 Jps
5 27285 ResourceManager
6 26695 NameNode
7 27082 SecondaryNameNode
8 27420 NodeManager
```

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# TPs

## TP1 : MapReduce with Hadoop

Project MapReduce :

`/home/hduser/BigDataHadoopSpark/TPs/TP1/MapReduce`

Two projects, A java Project and a python project

```
MapReduce|
         |--pom.xml
         |--bin|
         |--python|--mapper.py
         |        |--reduce.py
         |--src|--hadoop|--WordCount.Java
         |--target|
         |--test|wordcount|--file01
                            |--file02
                  |books|--b0
                        |--b1
```

# TPs
## TP1 : MapReduce with Hadoop

Java project:

- create directory in hdfs `/user/hduser/input`
- copy the files of `MapReduce/test/wordcount` in `/user/hduser/input`
- generate Java project BigDataTP1

```
1 cd BigDataHadoopSpark/TPs/TP1/MapReduce
2 mvn package
```

- apply Java WordCount application
- check results

# TPs
## TP1 : MapReduce with Hadoop

Python project

- create directory in hdfs `/user/hduser/input/book`
- copy the files of `MapReduce/test/book` in `/user/hduser/input`
- apply Python WordCount application
- check results

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# TPs
## TP3 : Installation Spark

### Spark Installation: spark-2.4.0-bin-hadoop-2.7.tgz

#### Installation

```
1 > cd /home/hduser
2 > mkdir local ; cd local
3 > wget https://www.apache.org/dyn/closer.lua/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
4 > tar xvfz spark-2.4.0-bin-hadoop-2.7.7.tar.gz
5 > mv spark-2.4.0-bin-hadoop-2.7.7 spark
6 > export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
7 > export SPARK_HOME=/home/hduser/local/spark
8 > export PATH=$SPARK_HOME/bin:$PATH
9 > export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
```

# TPs

## TP3 : Installation PySpark

### Installation

```
1 > cd /home/hduser
2 > export SPARK_HOME=/home/hduser/local/spark
3 > export PATH=$SPARK_HOME/bin:$PATH
4 > export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
5 > export PYSPARK_PYTHON="path_to_python"
6 > pip install pyspark
7 > pip install findspark
8 > sbin/start-master.sh
9 > sbin/start-slave.sh spark://localhost:7077
```

# TPs

## TP3 : Installation PySpark

### test Spark shell

```
1  > spark-shell
2
3  Setting default log level to "WARN".
4  To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
5  Spark context Web UI available at http://localhost:4040
6  Spark context available as 'sc' (master = local[*], app id = local-1578948405576).
7  Spark session available as 'spark'.
8  Welcome to
9        ____              __
10      / __/__  ___ _____/ /__
11     _\ \/ _ \/ _ `/ __/  '_/
12    /___/ .__/\_,_/_/ /_/\_\   version 3.0.0-preview2
13       /_/
14
15  Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_92)
16  Type in expressions to have them evaluated.
17  Type :help for more information.
18
19  scala>
```

# TPs

## TP3 : Installation PySpark

### test Spark shell

```
> pyspark


Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
/work/irlin355_1/gratienj/BigData/local/spark/spark-3.0.0-preview2-bin-hadoop2.7/python/pyspark/
      context.py:219: DeprecationWarning: Support for Python 2 and Python 3 prior to version 3.6 is
      deprecated as of Spark 3.0. See also the plan for dropping Python 2 support at https://spark.
      apache.org/news/plan-for-dropping-python-2-support.html.
  DeprecationWarning)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
___/__/ .__/\_,_/_/ /_/\_\   version 3.0.0-preview2
   /_/

Using Python version 2.7.5 (default, Apr 11 2018 07:36:10)
SparkSession available as 'spark'.
```

# TPs

## TP3 : Test0 Test1 Test2

Test0 :

- create spark context
- create liste of integer
- partition list with spark
- print num of partions

Test1 :

- compute square of integer list
- print square list

Test2 :

- compute PI

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# TPs

## TP4 : Spark ML, Data processing

Test0 :

- load `TPs/data/iris.csv` file in Panda DataFrame
- create Spark DataFrame
- show 5 first lines
- select two columns
- print some statistics on Spark Data frame

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# TPs

## TP5 : Spark ML, Data processing

Spark ML :

- load `TPs/data/iris.csv` file in Panda DataFrame
- create Spark DataFrame
- create Pipeline to prepare date for machine learning
- compute a predicting model
- evaluate the predicting model

# Outline

1. Introduction
2. Hadoop
   - Introduction
   - Architecture
   - HDFS
   - Yarn
   - MapReduce
3. Spark
   - Introduction
   - Architecture and Ecosystem
   - Spark Modules : Core, SQL, Mlib, Streaming and GraphX
     - Spark Core
     - Spark SQL
     - Spark Mlib
     - Spark Streaming
     - Spark GraphX
4. TP
   - TP0 : Hadoop Installation

# TPs

## Project : Spark ML, Image processing

project :

- load Lena.jpg file
- develop a parallel median Filter in python with Spark