**Plan détaillé de préparation au projet big data :**

Table des matières

[I. Liens pour les recherches : 2](#_Toc157387652)

[II. Ingestion et création du data lake 2](#_Toc157387653)

[**Source de données externe** 2](#_Toc157387654)

[III. Nettoyage et validation des données 2](#_Toc157387655)

[IV. Traitement des données 2](#_Toc157387656)

[Construction d’application distribué : Spark- calcul distribué 2](#_Toc157387657)

[Développer des requêtes SQL et NoSQL pour traiter des données volumineuses 3](#_Toc157387658)

[Utilisation de SQL 3](#_Toc157387659)

[V. Stockage sur hdfs 3](#_Toc157387660)

[VI. Orchestration 3](#_Toc157387661)

[Airflow ou commandes sh 3](#_Toc157387662)

[VII. Résumé 4](#_Toc157387663)

// commandes sh c’est pour romaric et airflow c’est l’enseignant : choisir commandes sh

# Liens pour les recherches :

Usefull

<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>

<https://sparkbyexamples.com/spark/spark-read-options/n>

<https://spark.apache.org/docs/latest/api/scala/org/apache/spark/ContextAwareIterator.html?search=dataframe>

<https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html>

**Top**

[**https://spark.apache.org/docs/latest/sql-getting-started.html**](https://spark.apache.org/docs/latest/sql-getting-started.html)

**exemples de toutes les fonctions du dataframes , et jointures**

[**https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html**](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html)

**Second**

<https://spark.apache.org/docs/latest/>

**Spark programming**

<https://spark.apache.org/docs/latest/sql-programming-guide.html>

Complet

# Ingestion et création du data lake

Le data lake est sur une architecture de stockage distribué (lecture et

écriture) : **stocké sur hdfs , fixer les droits d’accès**

* Lister les sources

## **Source de données externe**

**Le datalake est alimente en continue** ( avec une architecture , orchestrateur , message queu ou équivalent ) : **airflow automatique**

# Nettoyage et validation des données

Les données nettoyées et validées sont stockées dans un output

Intermédiaire avec un format qui prend en charge le schéma

# Traitement des données

## Construction d’application distribué : Spark- calcul distribué

**L’application utilise aussi des optimisations de la distribution et de répartition**

**de la charge**

val dfPartitionned = df.repartition(200)

**select avec cast :**

df.select(expr( " potential >=50 " ) , expr(" cast(overall as int) ") , expr("potential") ).show(10)

**EC Optimisations spark**

**Numpartitions**

df= spark.read.option().csv()

val partitionedDF = df.repartition(200)

Create tamp view

df.createOrReplaceTempView(tableName)

créer le cache : df.cache()

libérer : spark.catalog.clearCache()

… autres optimisations du cours

**imports nécessaires**

import org.apache.spark.sql

import org.apache.spark.sql.types

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

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

import org.apache.spark.sql.\_

import org.apache.spark.sql.expressions.Window

**Join exemple  :**

val playersPotential = {

players15.selectExpr(

"sofifa\_id", "short\_name", "cast(overall as int) overall\_15", "cast(potential as int) potential\_15"

).join(players22.selectExpr(

"sofifa\_id", "short\_name", "cast(overall as int) overall\_22", "cast(potential as int) potential\_22"

), "sofifa\_id")

.withColumn("evolution", expr("overall\_22-overall\_15"))

.withColumn("potential\_vs\_actual", col("overall\_22")-col("potential\_15"))

}

**Creation de colonne**

Val foo = foo.withColumn( ‘status’ ,

Expr( ‘ CASE WHEN delay <=10 then ‘on-time’ else ‘on-delays’ ’ )

 )

## **Développer des requêtes SQL et NoSQL pour traiter des données volumineuses**

## **Utilisation de SQL**

**L’application utilise une user-defined function ou une expression en SQL**

L’application utilise le DDL pour définir un schéma

// Register the DataFrame as a SQL temporary view

Df.read()

Df.cache()

Val repartDf = Df.répartition(8)

df.createOrReplaceTempView("people")

val sqlDF = spark.sql("SELECT \* FROM people")

sqlDF.show()

spark.sql("SELECT short\_name , overall FROM player22 order by overall desc " ).show

**avec Built in function**

val sqldf= {spark.sql("""SELECT short\_name , overall, nationality\_name, rank

| from (select short\_name , overall,nationality\_name , RANK()

| OVER( PARTITION BY nationality\_name Order BY overall DESC ) as rank

| from players22

| )

| where rank>=3

| ORDER BY overall desc

| """

| )}

val sqldf= {spark.sql("SELECT short\_name , overall, nationality\_name, rank

From (SELECT short\_name , overall, nationality\_name, RANK()

OVER( PARTITION BY nationality\_name Order By overall DESC) as rank

FROM player22

)

where rank >=3

" ) }

# Stockage sur hdfs

Capturer les ls , et mettre dans le rapport : hdfs /user/ubuntu/

* Ubuntu/Datalake (tous les dataset )
* Cleandata
* Dataresult
  + Dataresult1
  + Dataresult2 …

# Orchestration

## Airflow ou commandes sh

Le pipeline contient plusieurs étapes et l’enchainement est automatisé via

orchestrateur

* Faire plusieurs jobs , pour valider

**Exemple d’orchestration avec sh**

#!/bin/bash

# Créer un répertoire dans HDFS

hdfs dfs -mkdir -p livres

# Télécharger le fichier 'dracula'

wget <http://www.textfiles.com/etext/FICTION/dracula>

# Mettre le fichier 'dracula' dans le répertoire 'livres' de HDFS

hdfs dfs -put -f ingestion/storage user/ubuntu/datalake

# Lister les fichiers dans le répertoire 'livres' de HDFS

hdfs dfs -ls livres

# Récupérer le fichier 'center\_earth' du répertoire 'livres' de HDFS

hdfs dfs -get livres/center\_earth

dans un fichier script.sh, puis le rendre exécutable avec la commande chmod +x script.sh. Ensuite, vous pouvez l’exécuter avec ./script.sh.

créer un fichier sh pour chaque job , et appeler dans l’orchestrateur principale

Démarrer l’orchestrateur de façon automatisé

crontab -e

0 0 \* \* \* /home/ubuntu/Desktop/romaric/script.sh

# Résumé

1- avec python, récupérer les souces de données dynamique ( api )

stocker en json par exemple , sur le disk

- lister les sources statiques

2- un job spark vas créer le datalake dans hdfs

(stocker toutes les sources de données recoltes dans un dossier dans Hdfs (le data lake ) , )

3- un job spark (validation des données ) qui vas nettoyer et filtrer les données ,

et stocker dans un output intermédiaire (dossier dans hdfs )

4- des plusieurs jobs spark( environ 2 suivant l'épreuve )

de traitement des données du datalake (dans hdfs)

et stockage des résultat de chaque job dans un autre dossier hdfs

Optimiser (utiliser le cache, nombre de partitions parallèles pour la charge )

Savoir utiliser sparksql

exécuter les jobs en .sh ,ou relancer en automatique avec airflow

traiter les données (compétence obligatoire)

5- tout documenter dans le rapport word , a zipper

capture des résultats hdfs de chaque output ,

et tout le code dans le dossier word qui sera zipper

**Consignes :**

- Big Data

o Durée : 6h

o Travail demandé : créer un projet Spark, avec collecte transformation et présentation des données

o Modalité de retour : un fichier Zip contenant le projet

o Important :

 Vous travaillerez sur une machine virtuelle Linux/Ubuntu, avec Virtual Box

 Vous aurez à rendre une archive Zip : entrainez-vous à créer un Zip sur cette machine virtuelle

 Attention, veillez à bien tout prendre dans le Zip, seul son contenu sera évalué, les morceaux manquants ne seront pas récupérés.

**Faire des captures**

* Rapport
* built in