Processando grandes conjuntos de dados de forma paralela e distribuída com Spark

- ## Conceitos iniciais sobre Spark

> **SparkContext**
>
> Responsável por gerenciar o acesso aos arquivos, solicitação de recursos

> **Cluster Manager**
>
> Gerenciador de processos do Spark

> **Woker Node**
>
> São as maquinas que irão executar os processos

- ## Instalação e execução do Spark

> **wget https://github.com/fivethirtyeight/data/blob/master/avengers/avengers.csv**
>
> Faz download do arquivo avengers.csv

> **spark-shell**
>
> Acessa o spark-shell

> **val insurance = spark.read.format("csv").option("sep",",").option("header","true").load("file:///home/everis/avengers.csv")**
>
> Importa/Le o arquivo avengers.csv e cria um dataframe

> **insurance.show(1, false)**
>
> Exibe 1 linha do dataframe

> **insurance.select("URL").show(1, false)**
>
> Seleciona apenas o campo URL do dataframe

> **pyspark**
>
> Acessa o shell do pyspark

> **insurance = spark.read.format("csv").option("sep",",").option("header","true").load("file:///home/everis/avengers.csv")**
>
> Le o importa o arquivo para ser usado no pyspark

> **insurance.show()**
>
> Mostra o dataframe

- ## Spark SQL

> **SELECT * FROM csv.'file:///home/everis/avengers.csv'**
>
> Lendo um arquivo com Spark SQL

> **df.printSchema()**
>
> Mostra a estrutura do dataframe

> **df.show(50, false)**
>
> Mostra os dados do dataframe

> **df.select("campo1","campo2").show()**
>
> Seleciona por campos

> **df.select($"campo1",$"campo2"+1).show()**
>
> Faz transformação nos campos

> **df.filter($"age" > 21).show()**
>
> Executa o filtro baseado em uma coluna

> **df.groupBy("age").count().show()**
>
> Faz um agrupamento/contagem

> **df.withColumn("new_column_name", col("old_column_name")).show()**
>
> Renomeia nome da coluna

> **df.withColumn("new_column_name", col("old_column_name")).cast("long").show()**
>
> Renomeia nome da coluna e converte para tipo longo

> **df.avg("age").show()**
>
> Tira media da coluna age

> **df.sum("sales").show()**
>
> Soma coluna sales

> **df.max("age").show()**
>
> Obtem valor maximo de idade

- Para trabalhar com SQL puramente é necesario criar um Temp View

> **df.createTempView("people")**
>
> Cria uma Temp View

> **df.createOrReplaceTempView("people")**
>
> Cria ou substitui uma Temp View

> **df.createGlobalTempView("people")**
>
> **df.createOrReplaceGlobalTempView("people")**
>
> Global Temp View é para ambientes compartilhados

> **spark.sql("SELECT * FROM people").show()**
>
> **spark.sql("SELECT * FROM global_temp.people").show()**
>
> Excuta consultas SQL sobre as TempView

> **df.createOrReplaceTempView("av")**
>
> val df = spark.sql("SELECT Appearances FROM av WHERE URL LIKE '%IRON%'").show(10, false)

- Não é necesario quando estamos no shell. Porem usa-se quando vai criar um spark-submit

> - Cria um Spark Context em Scala

```
import org.apache.spark;SparkContext

import org.apache.spark.SparkConf

val conf = new SparkConf().setAppName("Meu App Spark")

val sc = new SparkContext(conf)
```

- Cria Spark Session em Scala

```
import org.apache.spark;SparkSession

val spark = SparkSession

.builder()

.appName("Spark SQL")

.config("Configuração", "Valor da Configuração")

.getOrCreate()
```

- Carregando dados usando o Spark SQL

```
val dfJson = spark.read.json("file:///home/everis/avengers.csv")

val dfParquet =
spark.read.format("json").load("file:///home/spark/Downloads/people.parquet")

val peopleDFCsv =
spark.read.format("csv").option("sep",",").option("header","true").load("file:///home/s
park/Downloads/people.parquet")
```

- Lendo por JDBC

```
val jdbcDF = sp.read

.format("jdbc")

.option("url", "jdbc:postgresql:dbserver")

.option("dbtable", "schema.tablename")

.option("user", "username")

.option("password", "password")

.option("driver", "com.driver.MyDriver")

.load()
```