Cheat sheet # 1 per Spark

Documentazione online

http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package

Session e Context

```
// Attenzione! Queste variabili sono definite solo
// negli ambienti interattivi (Notebook, spark-shell)
spark //-> SparkSession
spark.sparkContext //-> SparkContext
sc //-> SparkContext (equivalente a spark.sparkContext)
```

RDD

```
val rdd1 = sc.parallelize(Seq(10,20,30))
val rdd2 = sc.parallelize(1 to 10000)
// quante partizioni?
rdd2.partitions.length
rdd2.getNumPartitions
// è possibile specificare esplicitamente il numero
// di partizioni
val rdd3 = sc.parallelize(1 to 10000, 4)
```

Dataset

http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Dataset

```
val ds = spark.range(1000) //-> Dataset[bigint]
```

DataFrame

```
// Da Spark 2.0 i DataFrame sono Dataset[Row]
// per la documentazione online la pagina è la stessa
val df1 = spark.range(1000).toDF("x")
val df2 = spark.range(1000)
      .map(x=>(x,x*x))
      .toDF("x","y")
val df3 = Seq(
      ("rosso",1),
      ("verde",2),
      ("blu",3)).toDF("colore","codice")
// lo schema:
df3.printSchema
df3.schema
// posso creare un DataFrame a partire da un RDD[Row]
// e uno schema esplicito
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
val data = sc.parallelize(Seq(
      Row("rosso",1),
      Row("verde",2),
      Row("blu",3)))
val schema = StructType(Seq(
      StructField("colore", StringType,false),
      StructField("codice", IntegerType,false)))
val df4 = spark.createDataFrame(data,schema)
// per accedere ai campi di una Row bisogna
// fare un cast
val nomeColore = df4.first.getString(0)
val codice = df4.first.getInt(1)
```

Metodi comuni

```
// se data è un RDD[T], un Dataset[T] o un DataFrame
// posso usare:
data.count //-> numero di elementi
data.first //-> primo elemento
data.take(10) //-> primi 10 elementi
data.collect //-> tutti gli elementi
             // Attenzione! Possono essere troppi!
```

```
// rdd e Dataset (non i DataFrame!)possono
// essere manipolati in modo simile ai
// container di Scala:
data.map(x=>x*2).reduce(_+_)

// Dataset e Dataframe hanno un rdd soggiacente:
data.rdd.getNumPartitions
```

Manipolazione DataFrame

```
df.select("colore","valore")
df.select($"colore",$"valore" as "val")
df.selectExpr("colore","valore*2 as v2")
df.withColumn("y", "x*x")
df.withColumn("cost", lit(42))
df.withColumn("date", to_date($"date", "yyyy.MM.dd"))
df.withColumn("line",
      regexp_replace($"line", "[^a-zA-Z]"," "))
df.filter("x>0").filter("name is not null")
df.orderBy("nome","cognome")
df.limit(800) // crea un nuovo dataframe
df.distinct
df.withColumn("c", explode(split($"line", " ")))
df1.join(df2,df1.col("c1") === df2.col("c2"))
df1.union(df2)
```

UDF

```
def pow3(x:Integer):Integer = x*x*x
val pow3udf = udf(pow3(_:Integer):Integer)
df.select($"x", pow3udf($"x").as("x^3"))
```

SQL

```
df.createOrReplaceTempView("table")
spark.sql("""
      select city,count(1)
      from earthquake
      where richter>0
      group by city
""")
```

Lettura CSV

```
val df = spark.read
    .option("inferSchema", true)
    .option("header", true)
    .csv("hdfs:///user/cloudera/spark/earthquake.csv")
```

Aggregatori

```
df.groupBy("city").count()
df.groupBy("city").agg(count("*").as("numero"))
df.groupBy("city").agg(
      count("*"),
      avg("richter"),
      max("richter"))
df.withColumn("avg-richter",
      avg($"richter").over(Window
            .orderBy("date")
            .rowsBetween(-10,10)))
```