Máster en Big Data

Tecnologías de Almacenamiento

5. Hands-On: Desarrollo MapReduce

Àlex Balló Vergés
2018-2019

Índice

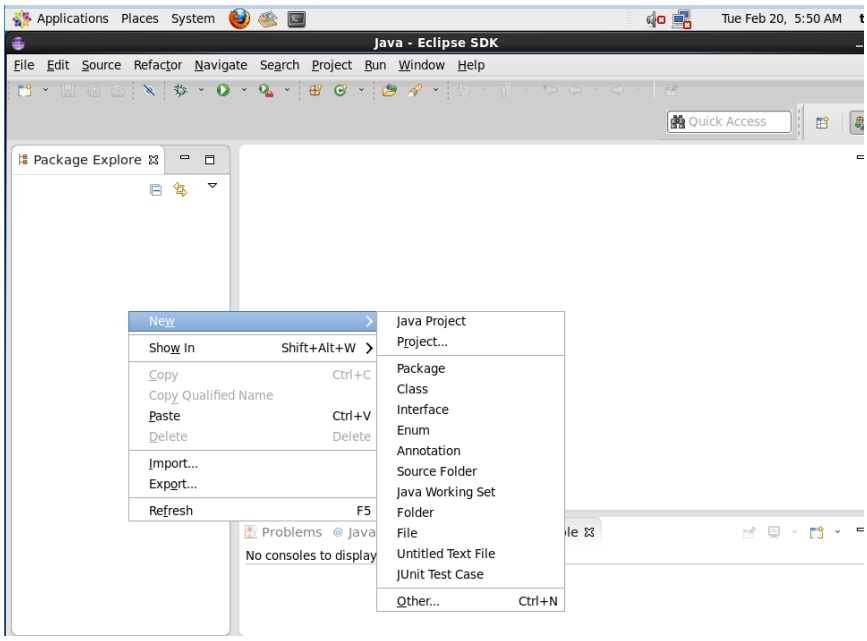
1. Introducción

El objetivo de este Hands-On es realizar el desarrollo, en Java, de dos Jobs sencillos de MapReduce y ejecutarlos.

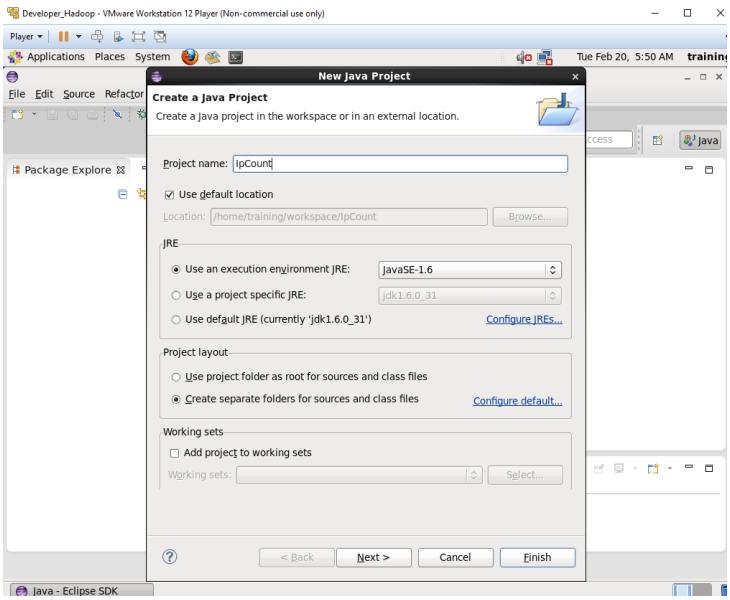
2. Entorno de desarrollo

Para realizar el desarrollo lo haremos mediante el IDE Eclipse de la máquina virtual importada en ejercicios anteriores.

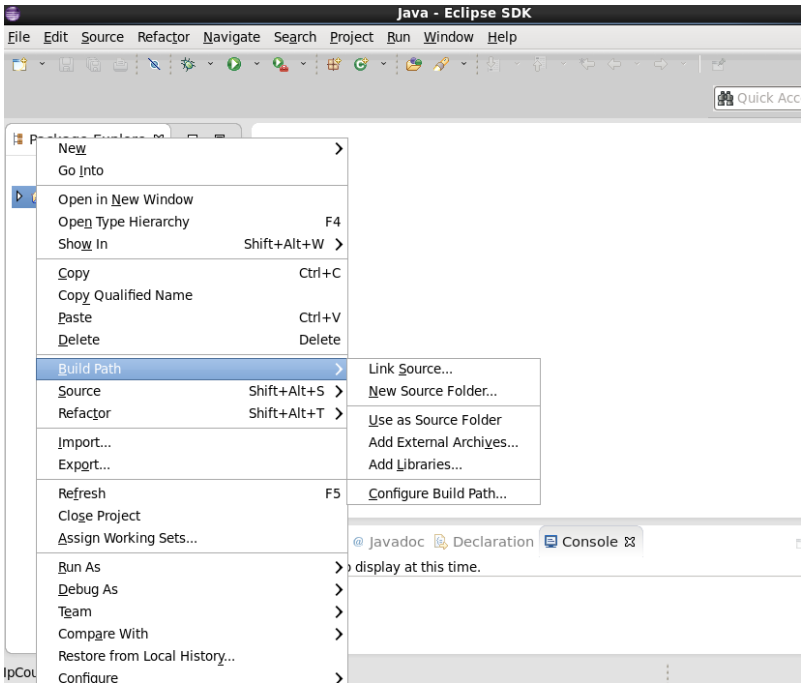
Para crear un nuevo proyecto, haremos click derecho sobre el package explorer New → Java Project



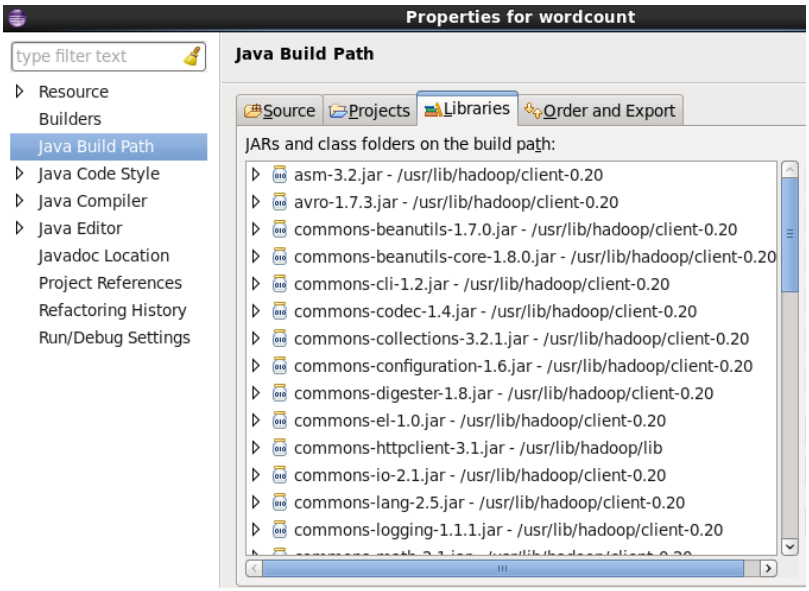
Introducimos el nombre del proyecto y click en Finish



Importamos manualmente las lilbrerías necesarias haciendo click derecho sobre el proyecto que acabamos de crear y seleccionamos Build Path → Configure Build Path



En la pestaña de libraries, seleccionamos Add Extertnal Jars e importamos todo el contenido de la carpeta /usr/lib/hadoop/client-0.20/



3. IpCount

Desarrollar y ejecutar el siguiente MapReduce:

Contar el número de veces que una misma Ip hace una petición en los logs contenidos en /weblog/ importado anteriorment

Código:

- Mapper:

```
public class IpMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
    @Override
    public void map (LongWritable  key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String ip=line.split(" ")[0];
        context.write( new Text(ip), new IntWritable(1));
    }

}
```

- Reducer:

```
public class CountReducer extends Reducer<Text,IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int ipCount=0;
        for (IntWritable value: values){
            ipCount += value.get();
        }
        context.write(key, new IntWritable(ipCount));
    }

}
```

- Controlador:

```
public class IpCount {
    public static void main(String[] args) throws Exception {

        if(args.length!=2){
            System.out.printf("Entrega la ruta de entrada y de salida");
            System.exit(-1);
        }

            Job job = new Job();
            job.setJarByClass(IpCount.class);
            job.setJobName("Contador de IPs");

            FileInputFormat.setInputPaths(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            job.setMapperClass(IpMapper.class);
            job.setReducerClass(CountReducer.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            boolean success = job.waitForCompletion(true);
            System.exit(success ? 0 : 1);


    }
}
```

Ejecución:

```
[training@localhost Jars]$ hadoop jar ipcount.jar ipcount.IpCount /user/training/weblog/ /user/training/ipcount
```

Resultado:

```
[training@localhost Jars]$ hadoop fs -cat /user/training/ipcount/part-r-00000 | head -n 50
10.1.1.113      1
10.1.1.125      12
10.1.1.144      1
10.1.1.195      4
10.1.1.236      12
10.1.1.5        1
10.1.10.155     2
10.1.10.197     2
10.1.10.198     1
10.1.10.48      1
10.1.10.5       1
10.1.100.104    1
10.1.100.13     1
10.1.100.138    1
10.1.100.183    14
10.1.100.199    35
10.1.100.5      1
10.1.101.135    29
```

4. Avarage Length

Desarrollar y ejecutar el siguiente MapReduce:

Calcular el promedio de la longitud de las palabras que empiecen por una letra determinada en todas las obras de Shakespeare importadas anteriormente

Ej de salida :

```
N 2.0
n 3.0
d 10.0
```

```
i 2.0
t 3.5
```

Código:

- Mapper:

```
public class AverageReducer extends Reducer<Text,IntWritable, Text, DoubleWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sumlen=0;
        int numberWords=0;
        for (IntWritable lengthword: values){
            numberWords++;
            sumlen += lengthword.get();
        }
        Double average=  (double) (sumlen/numberWords);
        context.write(key, new DoubleWritable(average));
    }

}
```

- Reducer:

```
public class AverageReducer extends Reducer<Text,IntWritable, Text, DoubleWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sumlen=0;
        int numberWords=0;
        for (IntWritable lengthword: values){
            numberWords++;
            sumlen += lengthword.get();
        }
        Double average=(double) sumlen/(double)numberWords;
        average= Math.round(average * 100.0) / 100.0;
        context.write(key, new DoubleWritable(average));
    }

}
```

- Controlador:

```
public class AverageLength {
    public static void main(String[] args) throws Exception {

        if(args.length!=2){
            System.out.printf("Entrega la ruta de entrada y de salida");
            System.exit(-1);
        }

            Job job = new Job();
            job.setJarByClass(AverageLength.class);
            job.setJobName("Media de longitud de palabras agrupados por letra inicial");

            FileInputFormat.setInputPaths(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            job.setMapperClass(AverageMapper.class);
            job.setReducerClass(AverageReducer.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(DoubleWritable.class);
            boolean success = job.waitForCompletion(true);
            System.exit(success ? 0 : 1);


    }
}
```

Ejecución:

```
[training@localhost Jars]$ hadoop jar averagelength.jar averageLength.AverageLength /user/training/shakespeare/ /user/training/averagelength
```

Resultado:

```
[training@localhost Jars]$ hadoop fs -cat /user/training/averagelength/part-r-00000
                5.16
&       3.33
'       4.93
(       8.71
-       8.0
.       1.0
1       3.5
2       2.75
4       3.0
5       3.0
6       3.0
8       2.5
:       1.0
A       7.39
B       8.93
C       9.71
D       6.86
E       8.34
F       7.94
G       8.97
H       7.21
I       2.07
J       6.77
K       5.52
L       7.71
M       8.62
N       7.55
O       5.2
P       9.35
```