IES SAN ANDRÉS

ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL Y BIG DATA

BIG DATA APLICADO

UT04: PROCESAMIENTO PARALELO EN APACHE HADOOP

ÍNDICE

1 PROCESAMIENTO DE DATOS CON MapReduce

1 PROCESAMIENTO DE DATOS CON MapReduce

1.1 EL MODELO MAPREDUCE

MODELO MAPREDUCE

Víctor J. González

MapReduce es un modelo de programación creado por Google en 2004 que permite procesar y generar grandes conjuntos de datos de manera distribuida.

Actualmente está en **desuso**, habiendo sido reemplazado por otras tecnologías como Spark.

MapReduce divide el procesamiento en dos fases principales, más una fase intermedia que realiza automáticamente:

- Fase MAP
- Fase SHUFFLE
- Fase REDUCE

Fase MAP

- Cada nodo del clúster recibe una parte o fragmento de los datos totales
- En esa parte, la función Map analiza los datos y produce una serie de pares clave-valor
- El tipo de análisis depende del problema

Ejemplo: contar veces que se repite cada palabra en un texto

```
'el gato come pescado'
'el perro come carne'
'el gato y el perro son amigos'
```

MAP

```
Nodo 1: ("el",1), ("gato",1), ("come",1),
("pescado",1)
Nodo 2: ("el",1), ("perro",1), ("come",1),
("carne",1)
Nodo 3: ("el",1), ("gato",1), ("y",1),
("el",1), ("perro",1), ("son",1),
("amigos",1)
```

Fase SHUFFLE & SORT

- El framework agrupa todos los pares clave-valor producidos en la fase Map por clave, reuniendo juntos todos los valores de una misma clave
- Se **ordenan** y distribuyen los pares por clave a los nodos encargados de la reducción

```
'el gato come pescado'
'el perro come carne'
'el gato y el perro son amigos'
```

MAP

```
Nodo 1: ("el",1), ("gato",1), ("come",1),
("pescado",1)
Nodo 2: ("el",1), ("perro",1), ("come",1),
("carne",1)
Nodo 3: ("el",1), ("gato",1), ("y",1),
("el",1), ("perro",1), ("son",1),
("amigos",1)
```

SHUFFLE

```
("amigos", [1])          ("perro", [1, 1])
("carne", [1])           ("pescado", [1])
("come", [1, 1])         ("son", [1])
("el", [1, 1, 1, 1])     ("y", [1])
("gato", [1, 1])
```

Fase REDUCE

- En la fase REDUCE, cada nodo recibe todos los valores que corresponden a una clave concreta
- La función Reduce toma cada clave y su conjunto de valores, y los procesa para producir un resultado final (por ejemplo, sumar el total para una palabra)
- El resultado final suele ser mucho más pequeño y se almacena normalmente en el sistema de archivos distribuido.

```
'el gato come pescado'
'el perro come carne'
'el gato y el perro son amigos'
```

MAP

```
Nodo 1: ("el",1), ("gato",1), ("come",1),
("pescado",1)
Nodo 2: ("el",1), ("perro",1), ("come",1),
("carne",1)
Nodo 3: ("el",1), ("gato",1), ("y",1),
("el",1), ("perro",1), ("son",1),
("amigos",1)
```

SHUFFLE

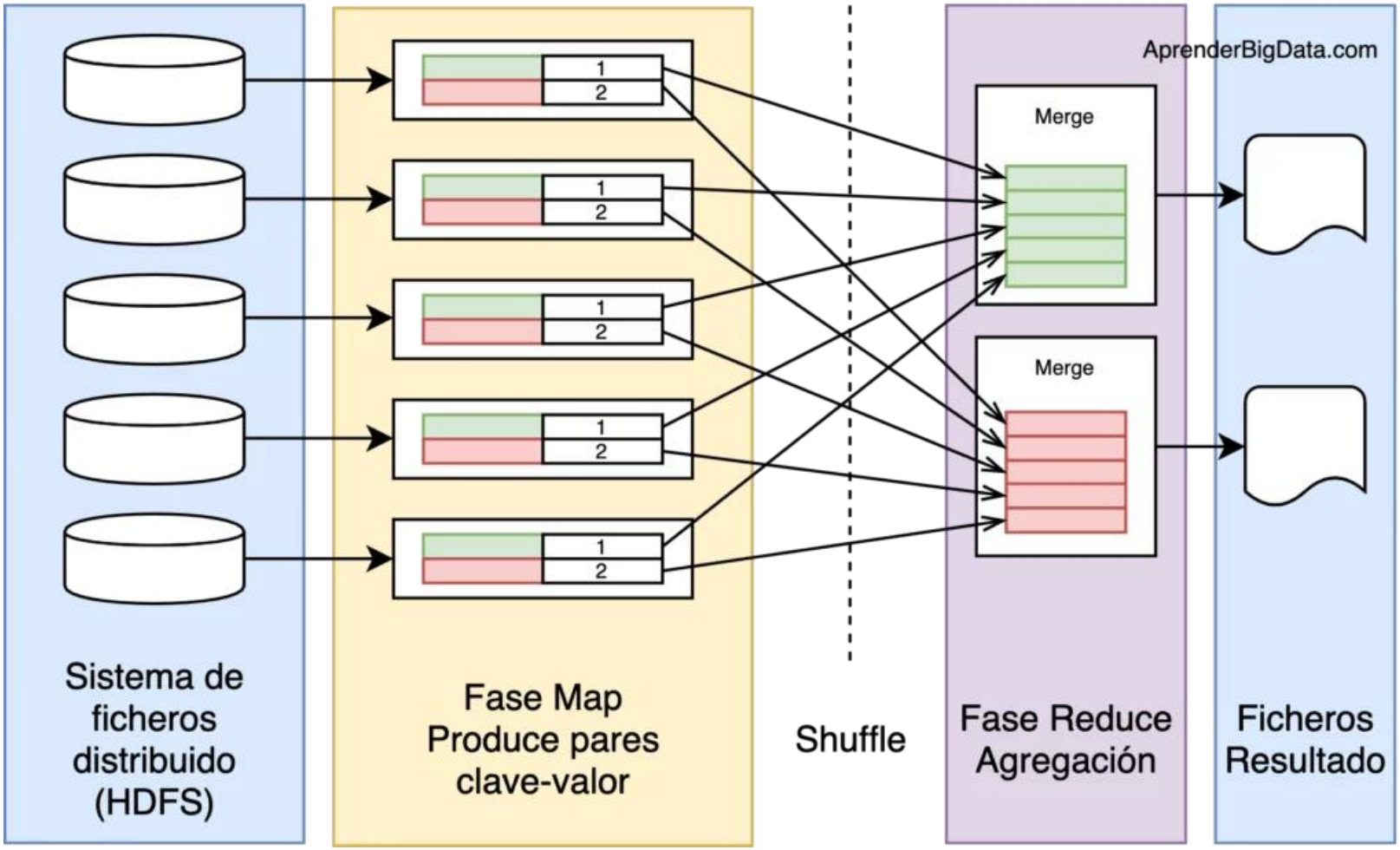
```
("amigos", [1])          ("perro", [1, 1])
("carne", [1])           ("pescado", [1])
("come", [1, 1])         ("son", [1])
("el", [1, 1, 1, 1])     ("y", [1])
("gato", [1, 1])
```

REDUCE

```
("amigos", 1)            ("perro", 2)
("carne", 1)             ("pescado", 1)
("come", 2)              ("son", 1)
("el", 4)                ("y", 1)
("gato", 2)
```

MODELO MAPREDUCE

Víctor J. González



1 PROCESAMIENTO DE DATOS CON MapReduce

1.2 CREACIÓN DE APLICACIÓN JAVA MAPREDUCE

PREPARANDO EL ENTORNO

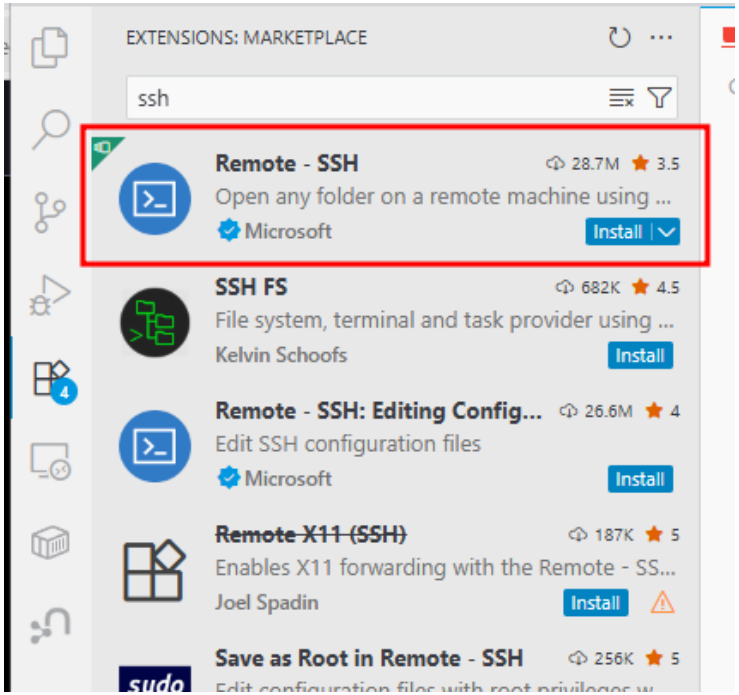
Víctor J. González

Vamos a crear ahora nuestra primera **aplicación Java** para MapReduce

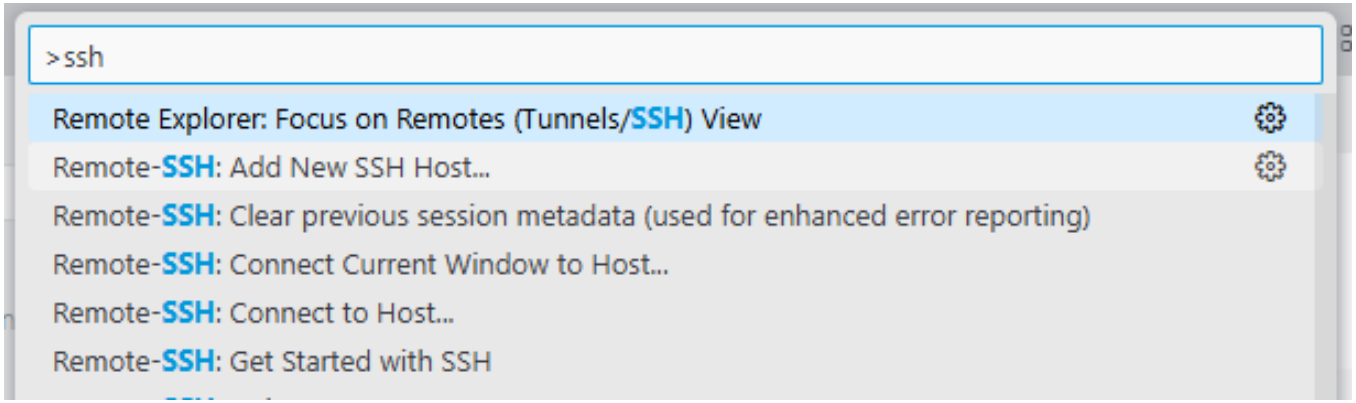
Tenemos que crear un fichero .java en el servidor Hadoop, dado que no tiene instalado ningún IDE ni entorno gráfico, lo más cómodo será trabajar de forma remota desde VS Code en nuestra propia máquina.

Para ello lo configuramos para conectarse de forma remota por SSH a otro equipo

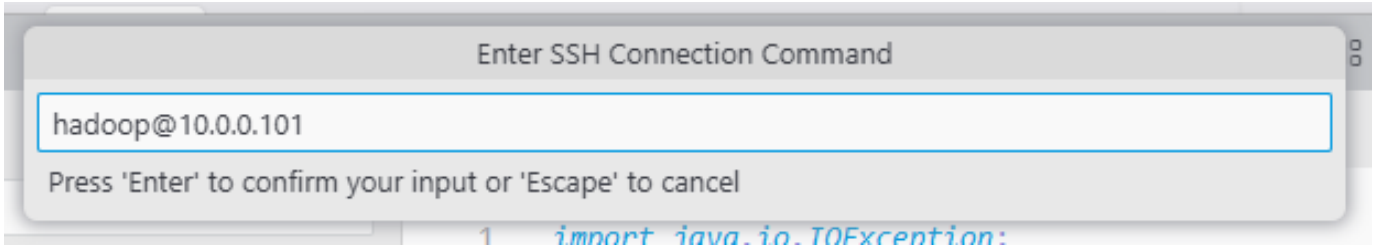
Comenzamos instalando la extensión **Remote - SSH**



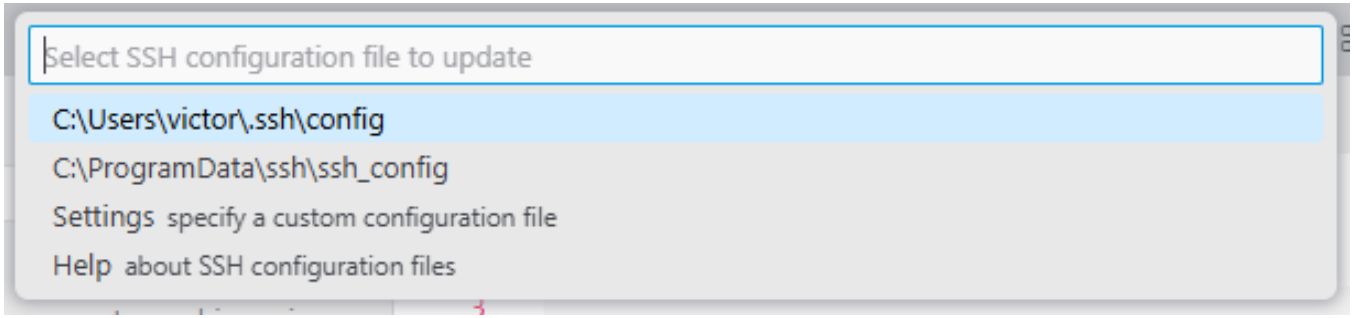
Una vez instalada, añadimos un nuevo host SSH



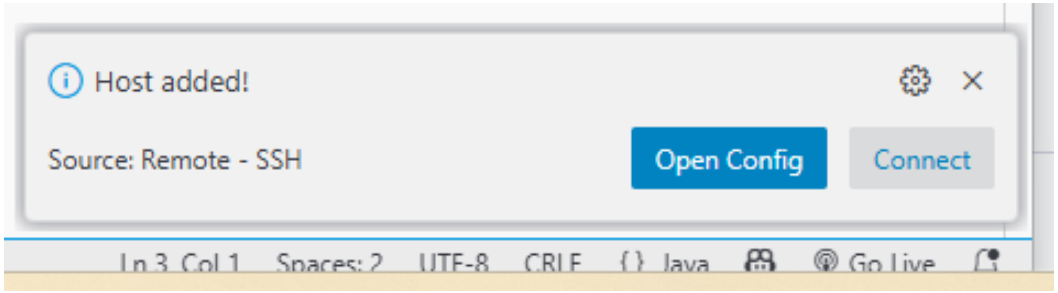
Le indicamos la IP del servidor con Hadoop y el nombre de usuario con el que estamos trabajando.



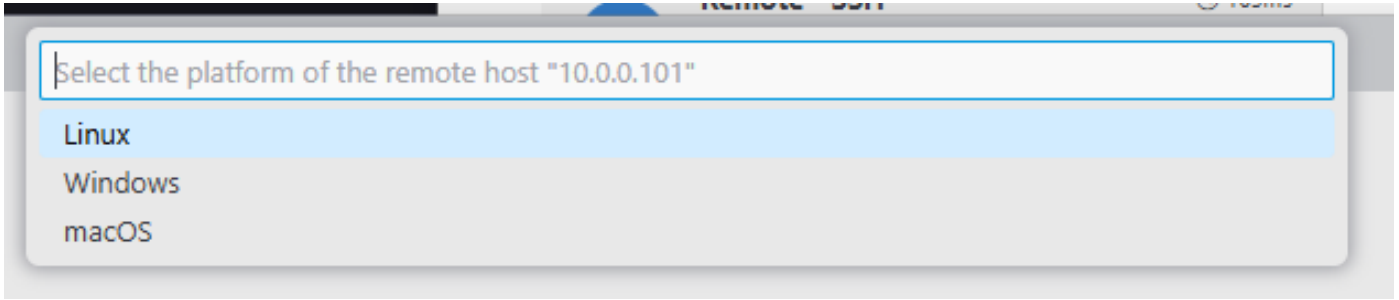
Indicamos el fichero de configuración SSH que queremos utilizar



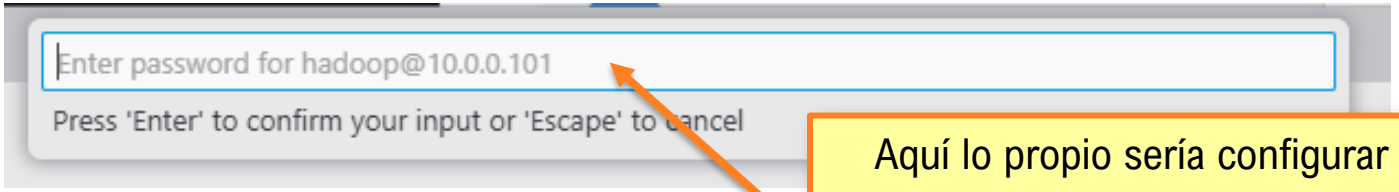
Y nos saldrá el mensaje de que se ha añadido con éxito



Le decimos que el sistema remoto es Linux

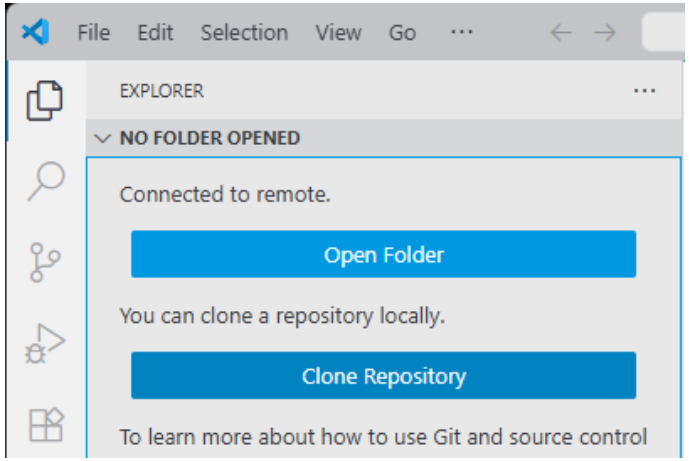


Y por último indicamos la clave de nuestro usuario

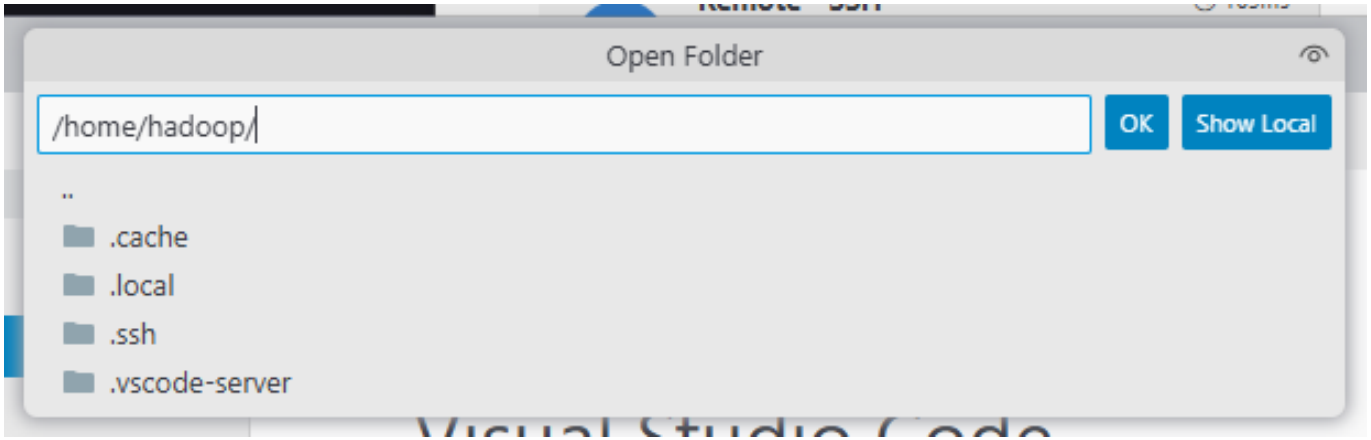


Aquí lo propio sería configurar el sistema para que se autentique mediante un par de claves pública-privada

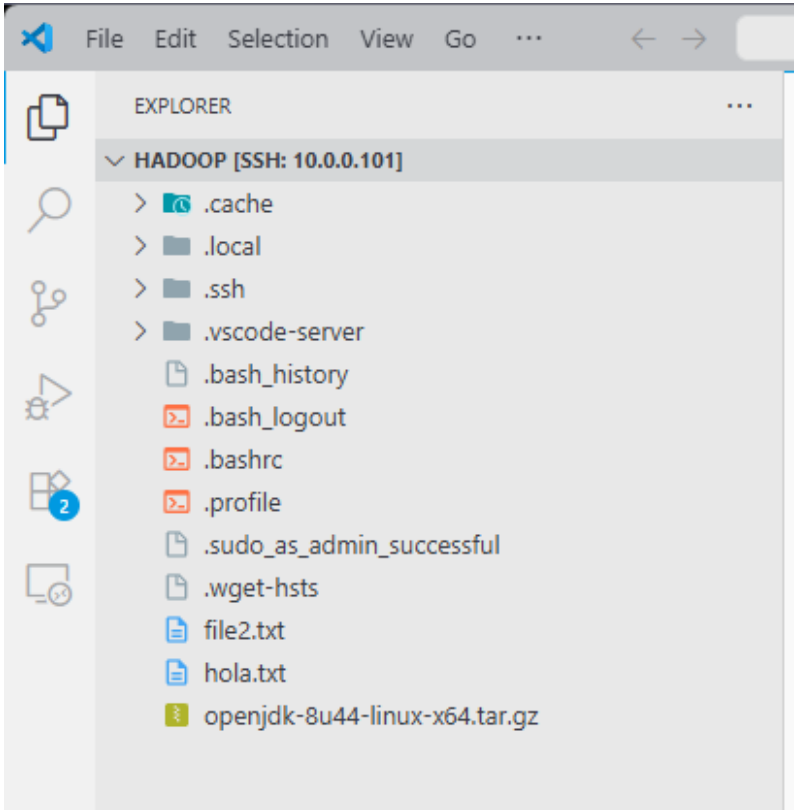
Ahora podemos ver que nos sale la opción de conectarnos a un directorio remoto

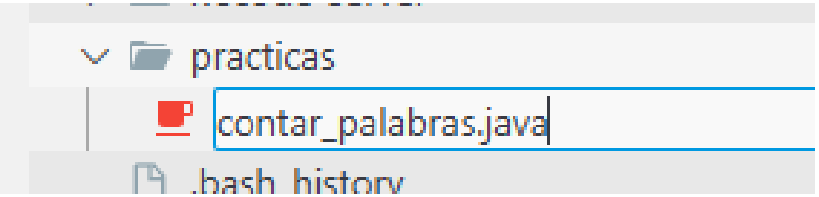


Seleccionamos el directorio donde queramos guardar nuestros programas Java para MapReduce



Y ya vemos el contenido del directorio

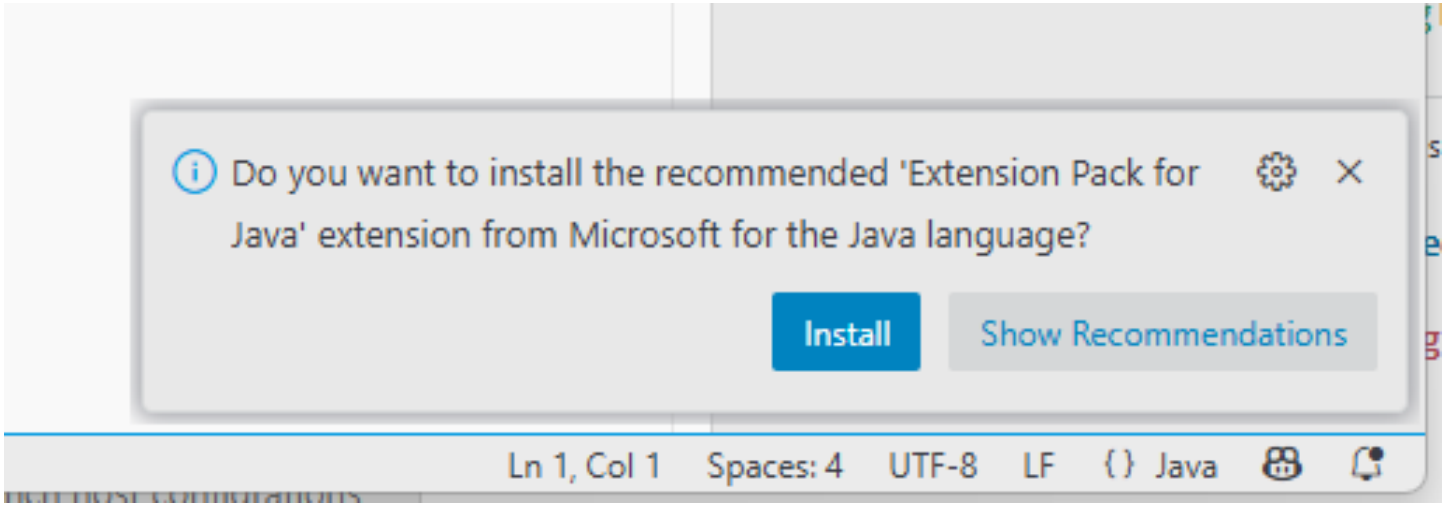




Vamos a crear un directorio que llamaremos `practicas` y dentro de él un fichero Java.

Como en este ejemplo crearemos un programa para contar palabras en un fichero, lo llamaremos `contar_palabras.java`

Cuando VS Code vea que estamos creando un fichero .java, nos dirá a ver si queremos instalar las extensiones para Java, le decimos que sí



APLICACIÓN JAVA

Víctor J. González

Para que una aplicación Java pueda ser utilizada por Hadoop necesita lo siguiente:

- Importar las clases de Hadoop
- Una clase que extienda el interfaz Mapper, que implementará el *mapper*
- Una clase que extienda el interfaz Reducer, que implementará el *reducer*
- Y una clase principal

IMPORTACIONES

Víctor J. González

Todas estas clases son de Hadoop

Lo usaremos para separar palabras en una línea de texto

Almacena la configuración de Hadoop (ficheros XML y parámetros del job)

Rutas dentro de HDFS o dentro del sistema de ficheros local

```
1   import java.io.IOException;
2   import java.util.StringTokenizer;
3
4   import org.apache.hadoop.conf.Configuration;
5   import org.apache.hadoop.fs.Path;
6   import org.apache.hadoop.io.IntWritable;
7   import org.apache.hadoop.io.Text;
8   import org.apache.hadoop.mapreduce.Job;
9   import org.apache.hadoop.mapreduce.Mapper;
10  import org.apache.hadoop.mapreduce.Reducer;
11  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13  import org.apache.hadoop.util.GenericOptionsParser;
```

Permite pasar parámetros opcionales al job

Definen cómo se leen y escriben los datos (entrada/salida de ficheros)

IMPORTACIONES

Víctor J. González

```
1  import java.io.IOException;
2  import java.util.StringTokenizer;
3
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.IntWritable;
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.mapreduce.Job;
9  import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13 import org.apache.hadoop.util.GenericOptionsParser;
```

Tipos de datos de Hadoop (no se usan tipos de datos de Java poque Hadoop necesita serialización eficiente)

Representa un trabajo de Hadoop

Clases base que se extienden para definir la lógica de MapReduce

LA CLASE MAPPER

Víctor J. González

Esta clase define el nombre de la aplicación. Aquí se agrupan el Mapper, el Reducer y el main

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19         extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

El Mapper transforma las líneas de entrada en pares clave-valor

LA CLASE MAPPER

Víctor J. González

Parámetros de tipo genérico que definen los tipos de entrada y salida para la clase Mapper. Indica a Hadoop qué tipo de datos manejará el mapeador.

org.apache.hadoop.mapreduce

Class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

```
16  pub
17
18    public static class TokenizerMapper
19        extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

LA CLASE MAPPER

Víctor J. González

Object: es el tipo de la clave de entrada, en el contexto de un formato de entrada de texto, la clave es el desplazamiento en bytes de la línea. Para mayor compatibilidad se define normalmente como `Object`.

```
16  pub
17
18    public static class TokenizerMapper
19        extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

LA CLASE MAPPER

Text: es el valor de la entrada, en este caso texto. La clase `Text` es la versión especializada y serializable de un `String` de Java que usa Hadoop

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19         extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

LA CLASE MAPPER

Víctor J. González

Text: el tipo para la clave de salida. Como vamos a contar palabras, la clave de salida será la palabra en sí

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19          extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

LA CLASE MAPPER

Víctor J. González

IntWritable: es el tipo para el valor de salida. Vamos a asignar un conteo de 1 a cada palabra que encontremos, por lo que el valor será un entero. `IntWritable` es el equivalente serializable de Hadoop de un `int` de Java.

```
16  pub
17
18    public static class TokenizerMapper
19        extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

LA CLASE MAPPER

Víctor J. González

`IntWritable`	entero de 32 bits (`int` en Java).
`LongWritable`	entero de 64 bits (`long`).
`FloatWritable`	número decimal de 32 bits (`float`).
`DoubleWritable`	número decimal de 64 bits (`double`).
`VIntWritable / VLongWritable`	enteros codificados con tamaño variable (ahorran espacio con números pequeños).
`Text`	cadena de caracteres UTF-8 (equivalente a `String`, optimizada para Hadoop).
`BooleanWritable`	valores `true`/`false`.
`ByteWritable`	un solo byte.
`BytesWritable`	array de bytes (útil para datos binarios, imágenes, etc.).
`ArrayWritable`	array de objetos `Writable`.
`MapWritable`	mapa clave–valor (`HashMap` serializable).
`SetWritable`	conjunto de objetos `Writable`.
`NullWritable`	“valor vacío”, se usa cuando no hace falta clave o valor
`ObjectWritable`	envoltorio genérico que puede contener cualquier objeto Java serializable.
`GenericWritable`	permite definir una lista de clases posibles para un campo.

LA CLASE MAPPER

Víctor J. González

Creamos un 1 de la clase `IntWritable`, es el valor que asignaremos a cada palabra

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19          extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

Variable que almacenará las palabras según las vayamos leyendo.

LA CLASE MAPPER

Víctor J. González

Siempre que creemos un Mapper, debemos implementar este método

Clave de entrada. En este ejemplo no lo vamos a utilizar

Valor de entrada, en este caso será el texto de entrada

```
16  public class ContarPalabras {
17
18    public static class Toke
19        extends Mapper<Obje
20
21      private final static IntWritable
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

LA CLASE MAPPER

Víctor J. González

```
16  public class ContarPalabras {
17
18    public static class TokenizerMappe
19        extends Mapper<Object, Text,
20
21      private final static IntWritable
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

Este es el objeto que permite enviar los resultados intermedios (pares <clave, valor>) al framework de Hadoop

LA CLASE MAPPER

Víctor J. González

```
16  public class ContarPalabras {
17
18    public static class TokenizerMappe
19        extends Mapper<Object, Text,
20
21      private final static IntWritable
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

La clase `StringTokenizer` de Java descompone una cadena en tokens. Genera un iterador que permite iterar sobre cada uno de los tokens

Convierte la entrada `value` en una cadena de Java

LA CLASE MAPPER

Víctor J. González

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19                 , Text, IntWritable>{
20
21      p          le one = new IntWritable(1);
22      p                 ;
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

Iteramos sobre cada uno de los tokens en que ha dividido la cadena. El bucle se ejecutará mientras queden más tokens

La función `nextToken()` devuelve el token correspondiente a esta iteración

LA CLASE MAPPER

Víctor J. González

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19         extends Mapper<Object, Text, Text,
20
21      private final static IntWritable one
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException, InterruptedException {
26        StringTokenizer itr = new StringTokenizer(value.toString());
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

Recuerda que habíamos creado una variable para almacenar cada una de las claves que devolvemos y que llamamos `word`

Usamos el método `set()` para establecer el valor de la variable

LA CLASE MAPPER

Víctor J. González

```
16  public class ContarPalabras {
17
18    public static class TokenizerMapper
19         extends Mapper<Object, Text, Text, IntWritable>{
20
21      private final static IntWritable one = new IntWritable(1);
22      private Text word = new Text();
23
24      public void map(Object key, Text value, Context context
25                      ) throws IOException
26        StringTokenizer itr = new StringTo
27        while (itr.hasMoreTokens()) {
28          word.set(itr.nextToken());
29          context.write(word, one);
30        }
31      }
32    }
33
```

Finalmente, guardamos el par `<clave, valor>` en el contexto indicando como clave la palabra y como valor 1

LA CLASE REDUCER

Víctor J. González

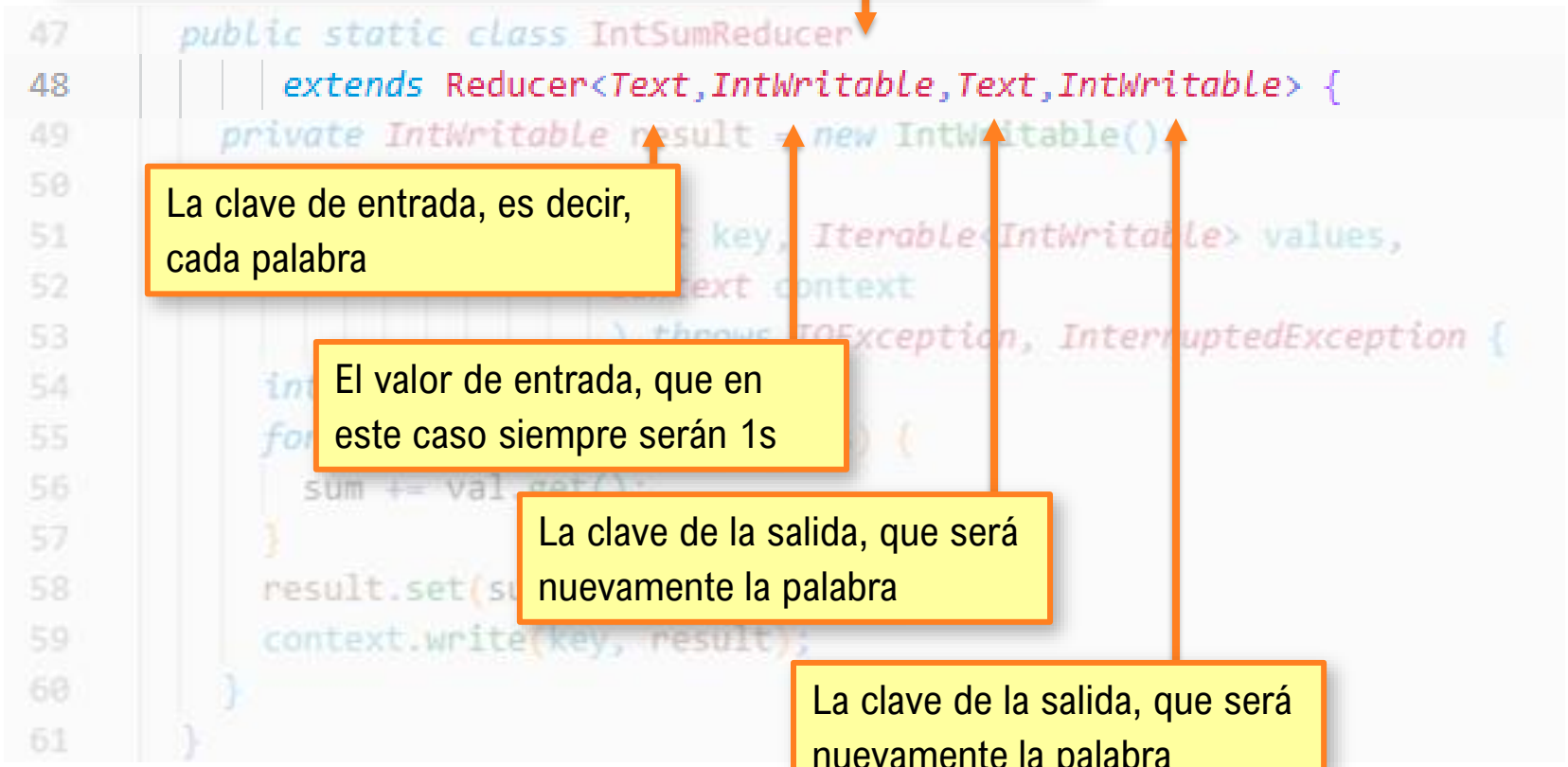
Ahora toca la clase para implementar el *reducer*, que heredará de la clase `Reducer`.

```
47  public static class IntSumReducer
48          extends Reducer<Text,IntWritable,Text,IntWritable> {
49    private IntWritable result = new IntWritable();
50
51    public void reduce(Text key, Iterable<IntWritable> values,
52                       Context context
53                       ) throws IOException, InterruptedException {
54      int sum = 0;
55      for (IntWritable val : values) {
56        sum += val.get();
57      }
58      result.set(sum);
59      context.write(key, result);
60    }
61  }
```

LA CLASE REDUCER

Víctor J. González

Empezamos definiendo los tipos de datos de la clave y valor de la entrada y la salida



La clave de entrada, es decir, cada palabra

El valor de entrada, que en este caso siempre serán 1s

La clave de la salida, que será nuevamente la palabra

La clave de la salida, que será nuevamente la palabra

LA CLASE REDUCER

Víctor J. González

```
47  public static class IntSumReducer
48       extends Reducer<Text,IntWritable,Text,IntWritable> {
49    private IntWritable result = new IntWritable();
50
51    public void reduce(Text key, Iterable<IntWritable> values,
52
53                                                    tedException {
54
55
56        sum += val.get();
57      }
58      result.set(sum);
59      context.write(key, result);
60    }
61  }
```

Declaramos una variable que utilizaremos para almacenar los resultados. Contendrá la suma de todos los 1s de una misma palabra

LA CLASE REDUCER

Víctor J. González

El método `reduce` es en el que tenemos que implementar el *reducer*.

```
47      public static class IntSumReducer
48              extends Reducer<Text,IntWritable,Text,IntWritable> {
49          private IntWritable result = new IntWritable();
50
51          public void reduce(Text key, Iterable<IntWritable> values,
52                             Context context
53                             ) throws IOException, InterruptedException {
54              int sum = 0;
55              for (IntWritable val : values) {
56                  sum += val.get();
57              }
58              result.set(sum);
59              context.write(key, result);
60          }
61      }
```

La palabra que corresponda

Nuevamente usaremos el contexto para devolver el resultado final

Este parámetro es un **iterable** con **todas las ocurrenciasn** que llegaron desde los *mappers* para esa palabra (en este ejemplo una secuencia de 1s). Es el **shuffle** quien ha realizado esta agrupación

LA CLASE REDUCER

Víctor J. González

Iteramos sobre todos los valores y los sumamos

```
47                                          cer
48          extends Reducer<Text,IntWritable,Text,IntWritable> {
49      private IntWritable result = new IntWritable();
50
51      public void reduce(Text key, Iterable<IntWritable> values,
52                         Context context
53                         ) throws IOException, InterruptedException {
54        int sum = 0;
55        for (IntWritable val : values) {
56          sum += val.get();
57        }
58        result.set(sum);
59        context.write(key, result);
60      }
61    }
```

Observa que la variable temporal `sum` es un entero de Java, por lo que puedo asignarle valor directamente con el operador =

Guardamos el resultado en la variable `result` (recuerda que era `IntWritable`) usando el método `set()`

LA CLASE REDUCER

Víctor J. González

```
47    public static class IntSumReducer
48        extends Reducer<Text,IntWritable,Text,IntWritable> {
49      private IntWritable result = new IntWritable();
50
51      public void reduce(Text key, Iterable<IntWritable> values,
52                         Context context
53                         ) throws IOException, InterruptedException {
54        int sum = 0;
55        for (IntWritable val : values) {
56          sum += val.get();
57        }
58        result.set(sum);
59        context.write(key, result);
60      }
61    }
```

Finalmente guardamos el par clave-valor en el contexto.

La clave es la palabra y el valor la suma que hemos realizado

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      Configuration conf = new Configuration();
68      String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();
69      if (otherArgs.length != 2) {
70        System.err.println("Usage: wordcount <in> <out>");
71        System.exit(2);
72      }
73      Job job = Job.getInstance(conf, "word count");
74      job.setJarByClass(ContarPalabras.class);
75      job.setMapperClass(TokenizerMapper.class);
76
77      job.setReducerClass(IntSumReducer.class);
78      job.setOutputKeyClass(Text.class);
79      job.setOutputValueClass(IntWritable.class);
80      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
81      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
83    }
84  }
```

Ya solo nos queda el programa principal que es el que se encarga de juntar todo y lanzar el trabajo

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      Configuration conf = new Configuration();
68      String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();
69      if (otherArgs.length != 2) {
70        System.err.println("Usage: wordcount <in> <out>");
71        System.exit(2);
72      }
73      Job job = new Job(conf, "word count");
74      job.setJarByClass(WordCount.class);
75      job.setMapperClass(TokenizerMapper.class);
76
77      job.setReducerClass(IntSumReducer.class);
78      job.setOutputKeyClass(Text.class);
79      job.setOutputValueClass(IntWritable.class);
80      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
81      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
83    }
84  }
```

Creamos un objeto Configuration que reúne la configuración de Hadoop.

Por defecto, carga los ficheros XML de configuración de Hadoop

EL PROGRAMA PRINCIPAL

Víctor J. González

Los argumentos que vamos a pasar cuando ejecutemos el programa

```
66      pu
67        Configuration conf = new Configuration();
68        String[] otherArgs = new GenericOptionsParser(conf, args).
          getRemainingArgs();
69        if (otherArgs.length != 2) {
70          System.err.println(x:"Uso: ContarPalabras <in> <out>");
71          System.exit(status:2);
72        }
73        Job job = Job.getInstanc
74        job.setJarByClass(Contar
75        job.setMapperClass(Token
76
77        job.setReducerClass(IntSumReducer.class);
78        job.setOutputKeyClass(Text.class);
79                                          ble.class);
80                                  new Path(otherArgs[0]));
81                                  ob, new Path(otherArgs[1]));
82                                  n(true) ? 0 : 1);
83      }
84    }
```

GenericOptionsParser procesa las opciones genéricas de Hadoop (p.e. `-D mapreduce.job.reduces=2`) y las aplica a `conf`

Devuelve los **argumentos no genéricos** (los que espera tu aplicación, por ejemplo, *input* y *output*)

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      Configuration conf = new Configuration();
68      String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();
69      if (otherArgs.length != 2) {
70        System.err.println(x:"Uso: ContarPalabras <in> <out>");
71        System.exit(status:2);
72      }
73      Job job = Job.getInstance(conf, "word count");
74      job.setJarByClass(ContarPalabras.class);
75      job.setMapperCl
76
77      job.setReducerCl
78      job.setOutputKey
79      job.setOutputValueClass(IntWritable.class);
80      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
81      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
83    }
84  }
```

Si el usuario no ha pasado dos argumentos (entrada y salida) finalizamos el programa.

EL PROGRAMA PRINCIPAL

Víctor J. González

La clase `Job` encapsula toda la información necesaria para ejecutar el trabajo en el clúster

Creamos una instancia con la configuración

Le asignamos un nombre que es el que aparecerá en la UI de YARN

EL PROGRAMA PRINCIPAL

Víctor J. González

Indica a Hadoop qué JAR debe distribuir a los nodos. Hadoop localiza el JAR que contiene la clase `ContarPalabras` y lo sube al HDFS/YARN para que los nodos ejecutores tengan el código.

```
74    job.setJarByClass(ContarPalabras.class);
75    job.setMapperClass(TokenizerMapper.class);
```

Asigna la clase `Mapper` que implementa la fase `map` (en nuestro ejemplo `TokenizerMapper`)

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      n();
68      nsParser(conf, args).
69      if (otherArgs.length != 2) {
70        System.err.println(x:"Uso: ContarPalabras <in> <out>");
71        System.exit(status:2);
72      }
73      Job job = Job.getInstance(conf, "word count");
74      job.setJarByClass(ContarPalabras.class);
75      job.setMapperClass(TokenizerMapper.class);
76
77      job.setReducerClass(IntSumReducer.class);
78      job.setOutputKeyClass(Text.class);
79      job.setOutputValueClass(IntWritable.class);
80      FileInputFormat.addInputPath(job, new Path(o
81      FileOutputFormat.setOutputPath(job, new Path
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
83    }
84  }
```

Indicamos cuál es la clase que implementa el *reducer* (en nuestro ejemplo `IntSumReducer`)

Establecemos los tipos de datos (Writable) que producirá el *job* en su salida final.

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      Configuration conf = new Configuration();
68      String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();
69
70
71        System.exit(status:2);
72      }
73      Job job = Job.getInstance(conf
74      job.setJarByClass(ContarPalabr
75      job.setMapperClass(TokenizerMa
76
77      job.setReducerClass(IntSumReducer.class);
78      job.setOutputKeyClass(Text.class);
79      job.setOutputValueClass(IntWritable.class);
80      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
81      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
83
84
```

Indicamos la ruta de entrada de los datos

`new Path()` crea rutas que pueden ser de HDFS (p.e. `hdfs://namenode:9000/user/algo`) o locales

La ruta de entrada puede ser un fichero o un directorio, además, admite comodines

Recogemos la ruta del primer parámetro que ha pasado el usuario.

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      Configuration conf = new Configuration();
68      String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();
69
70
71        System.exit(status:2);
72      }
73      Job job = Job.getInstance(conf, "word count");
74      job.setJarByClass(ContarPalabras.class);
75      job.setMapperClass(TokenizerMapper.class);
76
77      job.setReducerClass(IntSumReducer.class);
78      job.setOutputKeyClass(Text.class);
79      job.setOutputValueClass(IntWritable.class);
80      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
81      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
```

Indicamos la ruta de salida de los datos

NOTA: la ruta de salida **no debe existir** previamente en HDFS, si existiera el trabajo fallará

Recogemos la ruta del segundo parámetro que ha pasado el usuario.

EL PROGRAMA PRINCIPAL

Víctor J. González

```
66    public static void main(String[] args) throws Exception {
67      Configuration conf = new Configuration();
68      String[] otherArgs = new GenericOptionsParser(conf, args).
        getRemainingArgs();
69      if (
70        Sys                                    tarPalabras <in> <out>");
71        Sys
72      }
73      Job job = Job.getInstance(conf,
74      job.setJarByClass(Conta Palabras
75      job.setMapperClass(Toke izerMapp
76
77      job.setReducerClass(Int umReducer.class);
78      job.setOutputKeyClass(T ext.class);
79      job.setOutputValueClass( IntWritable.class);
80      FileInputFormat.addInput Path(job, new Path(otherArgs[0]));
81      FileOutputFormat.setOut utPath(job, new Path(otherArgs[1]));
82      System.exit(job.waitForCompletion(true) ? 0 : 1);
83    }
84  }
```

Enviamos el job al clúster y **bloque** hasta que termina

El valor true en el parámetro indica que se muestre el **progreso** en la consola.

Sale del programa con código 0 (éxito) o 1 (error).

COMPILACIÓN DEL PROGRAMA

Ahora que ya tenemos nuestro programa Java creado es el momento de **compilarlo** y ejecutarlo.

```
hadoop@hdfs-server:~/practicas$ ls -l
total 4
-rw-rw-r-- 1 hadoop hadoop 2339 ago 27 06:36 ContarPalabras.java
```

Para tener nuestro código ordenado, vamos a crear un subdirectorio para almacenar nuestro programa, en mi caso lo he llamado `wordcount`.

```
hadoop@hdfs-server:~/practicas$ mkdir wordcount
hadoop@hdfs-server:~/practicas$ mv ContarPalabras.java ./wordcount/
hadoop@hdfs-server:~/practicas$ ls -l ./wordcount/
total 4
-rw-rw-r-- 1 hadoop hadoop 2339 ago 27 06:36 ContarPalabras.java
```

COMPILACIÓN DEL PROGRAMA

Víctor J. González

Vamos a necesitar las clases de Haddop en el **classpath** para compilarlo.

Hadoop proporciona un script que nos da el classpath completo.

```
hadoop@hdfs-server:~/practicas$ hadoop classpath
/opt/hadoop/etc/hadoop:/opt/hadoop/share/hadoop/common/lib/*:/opt/hadoop/share/hado
doop/share/hadoop/hdfs:/opt/hadoop/share/hadoop/hdfs/lib/*:/opt/hadoop/share/hadoop
/share/hadoop/mapreduce/*:/opt/hadoop/share/hadoop/yarn:/opt/hadoop/share/hadoop/ya
p/share/hadoop/yarn/*
hadoop@hdfs-server:~/practicas$
```

COMPILACIÓN DEL PROGRAMA

Víctor J. González

Compilamos con **javac**

```
hadoop@hdfs-server:~/practicas/wordcount$ javac \
> -classpath `hadoop classpath` \
> -d . \
> ContarPalabras.java
```

Añadimos todas las librerías de Hadoop.

Las comillas inversas en Linux permiten ejecutar un subshell que ejecuta lo que haya entre comillas y lo reemplaza por la salida.

Esto es equivalente a escribir la salida de este comando.

Le indicamos que coloque los `.class` en el directorio actual.

Por último, el nombre de nuestra aplicación Java

Nos generará estos ficheros

```
hadoop@hdfs-server:~/practicas/wordcount$ ls -l
total 16
-rw-rw-r-- 1 hadoop hadoop 1754 ago 28 06:37 'ContarPalabras$IntSumReducer.class'
-rw-rw-r-- 1 hadoop hadoop 1751 ago 28 06:37 'ContarPalabras$TokenizerMapper.class'
-rw-rw-r-- 1 hadoop hadoop 1850 ago 28 06:37  ContarPalabras.class
-rw-rw-r-- 1 hadoop hadoop 2339 ago 27 06:36  ContarPalabras.java
```

COMPILACIÓN DEL PROGRAMA

Víctor J. González

Ahora empaquetamos los `.class` en un **JAR**

```
hadoop@hdfs-server:~/practicas/wordcount$ jar \
> cf \
> ContarPalabras.jar \
> *.class
```

Parámetro c (create): indica que queremos crear un nuevo fichero JAR. Si el archivo ya existe lo sobreescribe

Parámetro f (file): Indica que el nombre del fichero JAR viene a continuación. Si no indicamos este parámetro escribirá el resultado por la salida estándar (pantalla)

Nombre del fichero JAR

Ficheros que voy a meter dentro del JAR

EJECUCIÓN DEL PROGRAMA

Ya es el momento de **ejecutar el programa**.

Comenzamos creando un fichero con datos de prueba y copiándolo al sistema de ficheros HDFS

```
hadoop@hdfs-server:~/practicas/wordcount$ echo "hola mundo hola hadoop" > input.txt
hadoop@hdfs-server:~/practicas/wordcount$ hdfs dfs \
> -mkdir -p /user/hadoop/input
hadoop@hdfs-server:~/practicas/wordcount$ hdfs dfs \
> -put input.txt /user/hadoop/input
```

Podemos comprobar que se ha copiado correctamente

```
hadoop@hdfs-server:~/practicas/wordcount$ hdfs dfs -ls /user/hadoop/input
Found 1 items
-rw-r--r--   1 hadoop supergroup         23 2025-08-28 06:50 /user/hadoop/input/input.txt
```

EJECUCIÓN DEL PROGRAMA

Víctor J. González

Ya es el momento de **ejecutar el programa**.

```
hadoop@hdfs-server:~/practicas/wordcount$ hadoop
> ContarPalabras.jar \
> ContarPalabras \
> /user/hadoop/input \
> /user/hadoop/output
```

Archivo JAR que contiene el programa.

Internamente Hadoop subirá este archivo al directorio de **staging** para que los contenedores lo usen

Nombre de la clase principal

Ruta del fichero de entrada y del de salida.

Sin prefijo explícito (`hdfs://` o `file://`), esa ruta se interpreta según `fs.defaultFS` en el fichero `core-site.xml`

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:8020</value>
  <final>false</final>
  <source>core-site.xml</source>
</property>
```

Importante: el directorio de salida no debe existir, si existiera el trabajo fallará

EJECUCIÓN DEL PROGRAMA

Al finalizar la ejecución podremos comprobar el directorio de salida.

Veremos que ha creado un fichero `_SUCCESS` vacío para indicar que el programa ha finalizado con éxito y un fichero `part-r-00000` con la salida del programa

```
hadoop@hdfs-server:~/practicas/wordcount$ hdfs dfs -ls /user/hadoop/output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2025-08-28 06:53 /user/hadoop/output/_SUCCESS
-rw-r--r--   1 hadoop supergroup         24 2025-08-28 06:53 /user/hadoop/output/part-r-00000
```

Si vemos el contenido de este fichero tendremos el resultado de la ejecución del programa

```
hadoop@hdfs-server:~/practicas/wordcount$ hdfs dfs -cat /user/hadoop/output/part-r-00000
hadoop  1
hola    2
mundo   1
```

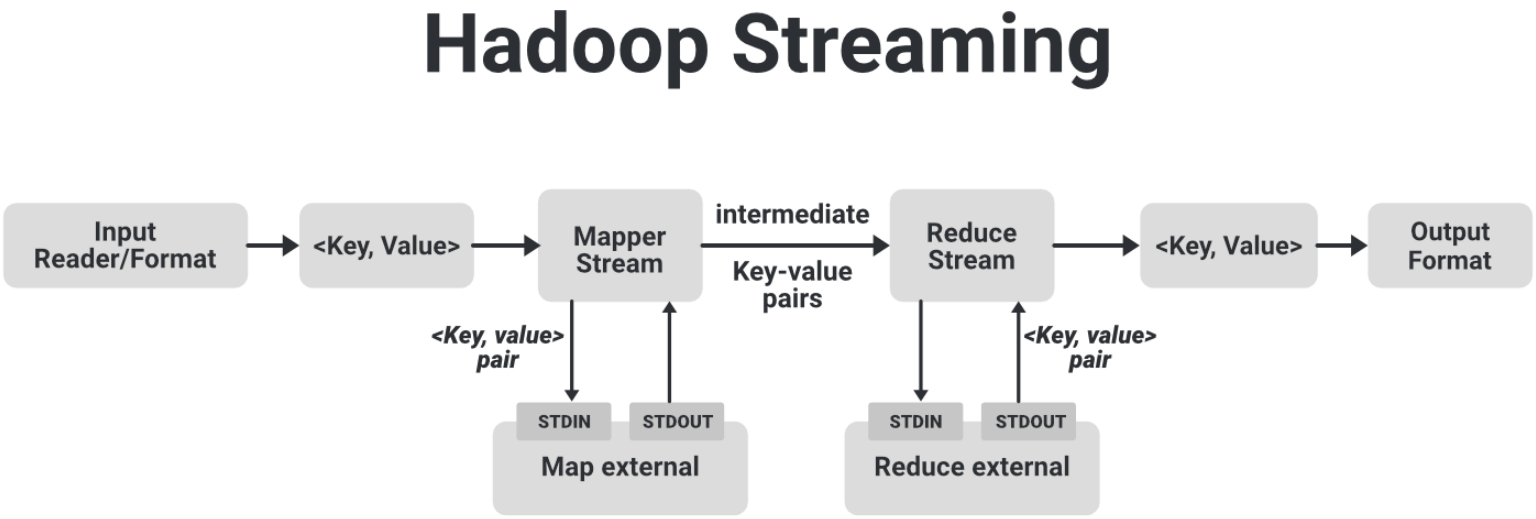
2 MAPREDUCE CON STREAMING PYTHON

2 MAPREDUCE CON STREAMING PYTHON

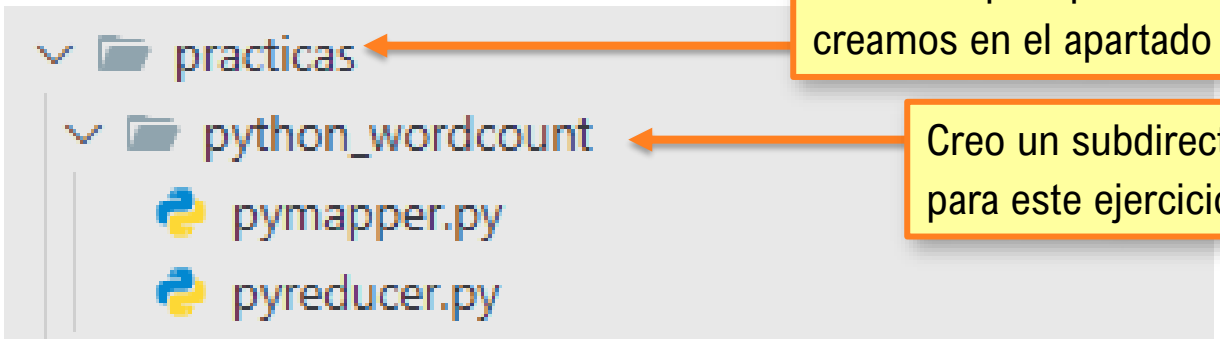
2.1 EJEMPLO DE CONTAR PALABRAS EN PYTHON

Hadoop Streaming es una utilidad que permite ejecutar trabajos MapReduce con programas en cualquier lenguaje que pueda leer de la entrada estándar y escribir en la salida estándar.

Esto significa que podemos usar scripts en Python como *mapper* y *reducer*.



Ejemplo: Contar palabras



Voy a utilizar el mismo directorio para prácticas que creamos en el apartado anterior

Creo un subdirectorio para este ejercicio

Y dentro de él creo un fichero Python para el *mapper* y otro para el *reducer*.

Recuerda que los scripts de Python tienen extensión `.py`

El código del ***mapper*** será el siguiente:

```python
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

Esta es la **shebang line** obligatoria en todos los scripts de Linux que indica dónde está el programa que ejecutará el script

```
1  #!/usr/bin/env python3
2  import sys
3
4  for line in sys.stdin:
5      li
6      wo
7      fo
8
```

El programa `/usr/bin/env` está incluido en casi todas las distribuciones Linux y su función es buscar un ejecutable en el `PATH` y ejecutarlo

Necesitaremos la librería estándar de Python para acceder a `sys.stdin` y `sys.stdout`

¿Por qué usamos `/usr/bin/env python3` en lugar de `/usr/bin/python3`? Porque en algunas distribuciones Python puede estar en otras ubicaciones y así nos aseguramos de encontrarlo esté donde esté, haciendo el script más **portable**.

Recuerda que si quieres conocer la ubicación de un archivo en Linux siempre puedes utilizar el comando `which`

```
hadoop@hdfs-server:~$ which python3
/usr/bin/python3
```

HADOOP STREAMING

Víctor J. González

Hadoop pasa cada línea del fichero de entrada al script a través de la **entrada estándar**.

Este bucle procesa cada línea una a una, esto permite trabajar con archivos muy grandes sin cargarlos enteros en memoria

```
1
2   import sys
3
4   for line in sys.stdin:
5       line = line.strip()
6       words = line.split()
7       for word in words:
8           print(f"{word}\t1")
```

La función `str.strip()` elimina espacios en blanco al principio y al final, así como los saltos de línea `\n`

La función `str.split()` divide la línea en palabras separadas por espacios y las devuelve en un array.

Ejemplo: `"Hola mundo Hadoop" → ["Hola", "mundo", "Hadoop"]`

Iteramos sobre cada una de las palabras

```python
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

Hadoop espera que devolvamos cada par <clave, valor> en una línea separados por un tabulado.

Por ejemplo, para `'hola hola mundo'` espera lo siguiente:

```
hola    1
hola    1
mundo   1
```

Tenemos que devolver el resultado por la entrada estándar, por lo que utilizamos `print`

Y ahora el código del ***reducer***:

```python
#!/usr/bin/env python3
import sys

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.strip().split("\t", 1)
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word:
    print(f"{current_word}\t{current_count}")
```

```python
#!/usr/bin/env python3
import sys

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.strip().split('\t', 1)
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word:
    print(f"{current_word}\t{current_count}")
```

Aquí recibimos los datos después de la fase Shuffle&Sort que realiza Hadoop automáticamente.

La forma en que nos entregará los datos es ordenando todos los pares clave/valor que ha generado el *mapper* por clave, y devolviendo cada par clave/valor en una línea, con **todos los pares con la misma clave en líneas consecutivas**.

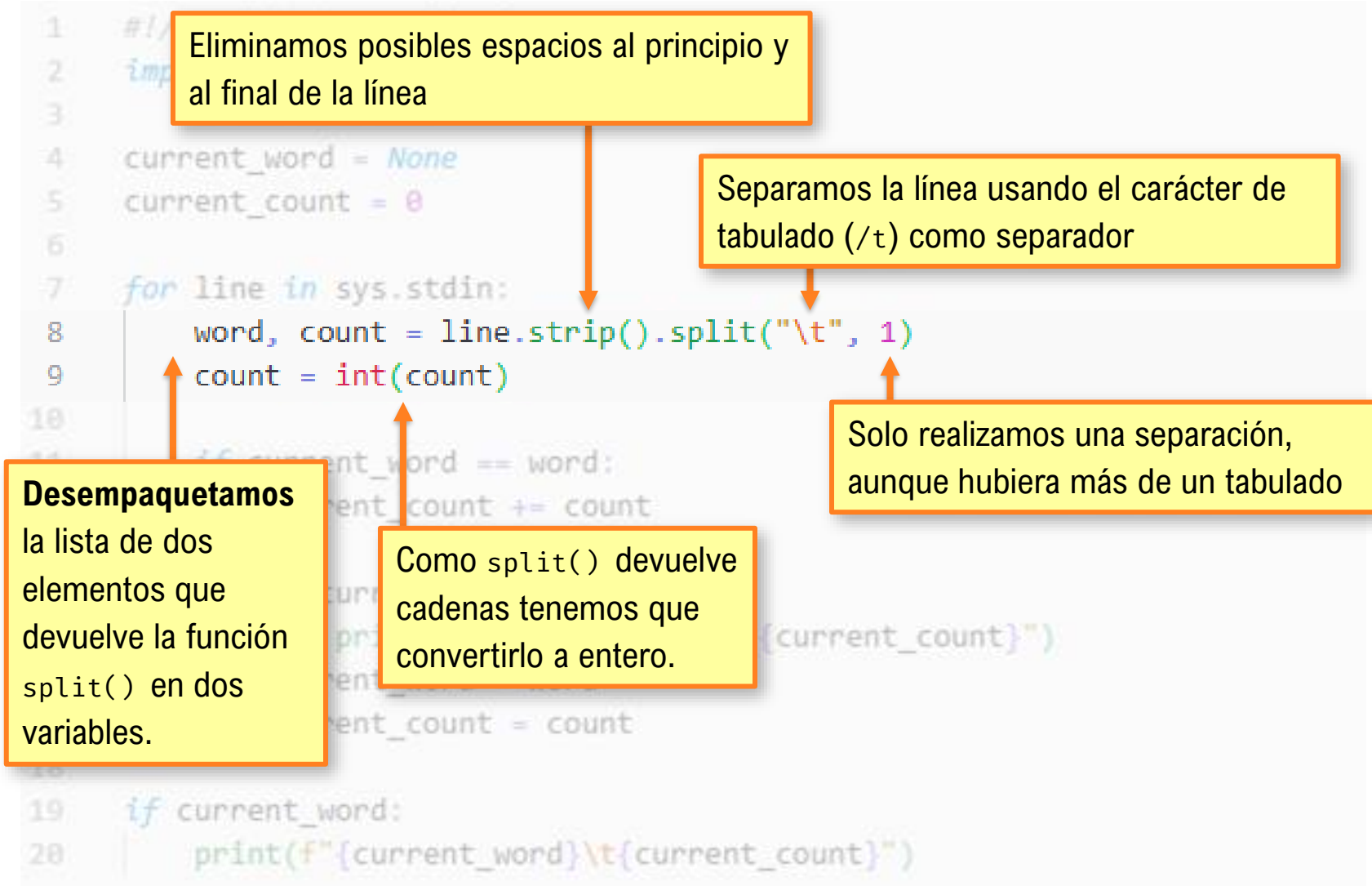
Se entregan al *reducer* a través de la entrada estándar

```python
#!/usr/bin/env python3
import sys

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.strip().split("\t", 1)
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word:
    print(f"{current_word}\t{current_count}")
```

Establecemos una variable para recordar la palabra actual y otra para el contador de veces que ha aparecido esa palabra

Leemos una a una todas las líneas de la entrada estándar.

Cada línea contendrá un par clave/valor separados por un tabulado


Eliminamos posibles espacios al principio y al final de la línea
Separamos la línea usando el carácter de tabulado (/t) como separador
word, count = line.strip().split("\t", 1)
count = int(count)
Desempaquetamos la lista de dos elementos que devuelve la función split() en dos variables.
Como split() devuelve cadenas tenemos que convertirlo a entero.
Solo realizamos una separación, aunque hubiera más de un tabulado

Comprobamos si la palabra es la misma que la de las líneas anteriores

Si es la misma, incrementamos el contador

Comprobamos que no sea null y enviamos a la salida estándar la palabra actual y el valor del contador

Recordamos la nueva palabra e inicializamos el contador

```
5   current_count = 0
6
7   for line in sys.stdin
8       word, count = lin                ("\t", 1)
9       count = int(count)
10
11      if current_word == word:
12          current_count += count
13      else:
14          if current_word:
15              print(f"{current_word}\t{current_count}")
16          current_word = word
17          current_count = count
18
19  if current_word:
20      print(f"{current_word}\t{current_count}")
```

Ya tenemos nuestra aplicación Python para ejecutar en Hadoop, por lo que es el momento de **ejecutarla**.

Antes de lanzar la aplicación sobre Hadoop, podemos probar nuestro código en un sistema Linux ya que, a fin de cuentas, el programa:

- Recoge los datos de la entrada estándar.
- La fase *shuffle* la podemos simular con el comando sort
- Los datos de salida se envían a la salida estándar

Preparo los datos y lanzo el mapper y el reducer

[5]:
```
%%writefile data
asir dam asir asir daw dam asir asir
```

```
Overwriting data
```

[6]:
```
!ls
```

```
Untitled.ipynb  data  mapper.py  reducer.py
```

[7]:
```
!cat data \
| python3 mapper.py \
| sort \
| python3 reducer.py
```

```
asir    5
dam     2
daw     1
```

Esta es la salida del proceso de simular el MapReduce

HADOOP STREAMING

Víctor J. González

Ahora que sabemos que nuestro código es correcto, es momento de lanzarlo sobre MapReduce.

En este caso tenemos que utilizar la utilidad **streaming** de Hadoop, que es la que permite procesar datos obtenido de la entrada estándar y volcarlos a la salida estándar.

Para ello usaremos la aplicación **hadoop-streaming**

```
hadoop@hdfs-server:~$ ls /opt/hadoop/share/hadoop/tools/lib/
aliyun-java-sdk-core-4.5.10.jar          hadoop-fs2img-3.4.1.jar
aliyun-java-sdk-kms-2.11.0.jar           hadoop-gridmix-3.4.1.jar
aliyun-java-sdk-ram-3.1.0.jar            hadoop-kafka-3.4.1.jar
aliyun-sdk-oss-3.13.2.jar                hadoop-minicluster-3.4.1.jar
azure-data-lake-store-sdk-2.3.9.jar      hadoop-resourceestimator-3.4.1.jar
azure-keyvault-core-1.0.0.jar            hadoop-rumen-3.4.1.jar
azure-storage-7.0.1.jar                  hadoop-sls-3.4.1.jar
bundle-2.24.6.jar                        hadoop-streaming-3.4.1.jar
hadoop-aliyun-3.4.1.jar                  hamcrest-core-1.3.jar
hadoop-archive-logs-3.4.1.jar            ini4j-0.5.4.jar
hadoop-archives-3.4.1.jar                jdk.tools-1.8.jar
hadoop-aws-3.4.1.jar                     jdom2-2.0.6.1.jar
```

Algo importante antes de lanzar Hadoop es asegurarnos de que nuestros scripts tengan permisos de ejecución.

Si no los tuvieran se los damos

```
hadoop@hdfs-server:~/practicas/python_wordcount$ ls -l
total 8
-rw-rw-r-- 1 hadoop hadoop 174 ago 29 06:52 pymapper.py
-rw-rw-r-- 1 hadoop hadoop 439 ago 30 06:13 pyreducer.py
hadoop@hdfs-server:~/practicas/python_wordcount$ chmod u+x *.py
hadoop@hdfs-server:~/practicas/python_wordcount$ ls -l
total 8
-rwxrw-r-- 1 hadoop hadoop 174 ago 29 06:52 pymapper.py
-rwxrw-r-- 1 hadoop hadoop 439 ago 30 06:13 pyreducer.py
```

Lo ejecutamos de la siguiente forma:

```
hadoop@hdfs-server:~$ hadoop jar \
> /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar \
> -file ~/practicas/python_wordcount/pymapper.py \
> -mapper ~/practicas/python_wordcount/pymapper.py \
> -file ~/practicas/python_wordcount/pyreducer.py \
> -reducer ~/practicas/python_wordcount/pyreducer.py \
> -input /user/hadoop/input/input.txt \
> -output /user/hadoop/output
```

Ejecutamos el JAR Hadoop Streaming que permite ejecutar programas externos como *mappers* y *reducers*

El parámetro `-file` indica que el archivo que se indica a continuación debe **enviarse desde el sistema local** al nodo del clúster donde se ejecuta el *mapper* o el *reducer*.

Este parámetro se asegura de que todos los nodos tengan una copia del script

Indicamos que script corresponde al *mapper* y cuál al *reducer*

Cuál será la fuente de **datos de entrada** (si fuera un directorio procesaría todos los ficheros que contiene) así como el **directorio de salida** (que no debe existir)

Estas rutas son dentro de HDFS

Si vamos al directorio que indicamos como salida veremos que es análoga al ejemplo que realizamos con Java

```
hadoop@hdfs-server:~$ hdfs dfs -ls /user/hadoop/output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2025-08-30 06:36 /user/hadoop/output/_SUCCESS
-rw-r--r--   1 hadoop supergroup         24 2025-08-30 06:36 /user/hadoop/output/part-00000
hadoop@hdfs-server:~$
```

Y podremos ver el resultado dentro del fichero `part-00000`

```
hadoop@hdfs-server:~$ hdfs dfs -cat /user/hadoop/output/part-00000
hadoop  1
hola    2
mundo   1
```

2 MAPREDUCE CON STREAMING PYTHON

2.2 USO DE JUPYTER NOTEBOOK CON HADOOP

JUPYTER NOTEBOOK

Víctor J. González

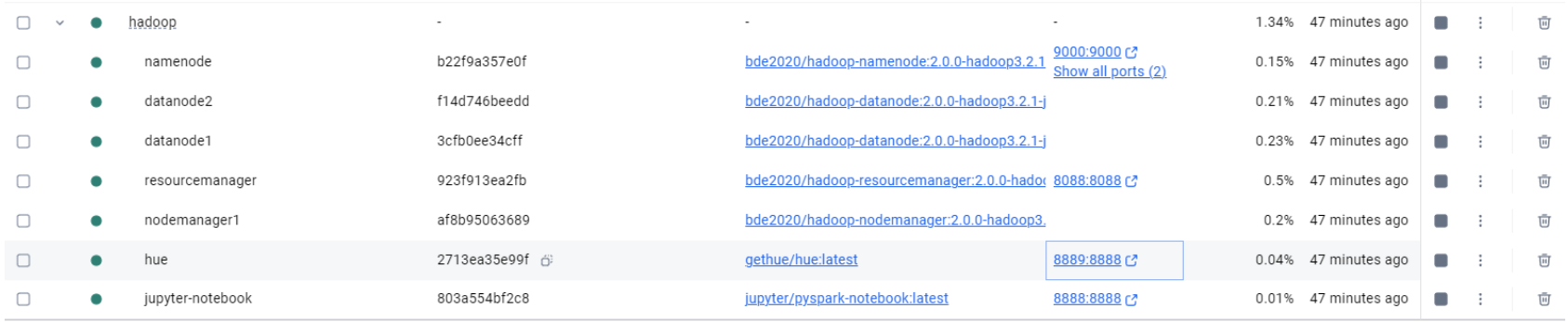
Vamos a ver ahora cómo interactuar con Hadoop utilizando **Jupyter Notebook**.

Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos que contienen **código vivo, resultados de ejecución, visualizaciones y documentación**.

Es el entorno ideal para ciencia de datos, ya que permite ejecutar código bloque a bloque y ver los resultados de inmediato.

Si tenemos Hadoop instalado sobre equipos físicos o máquinas virtuales, deberíamos instalarlo en alguna máquina del clúster o configurar la máquina que tenga Jupyter para conectarse al clúster.

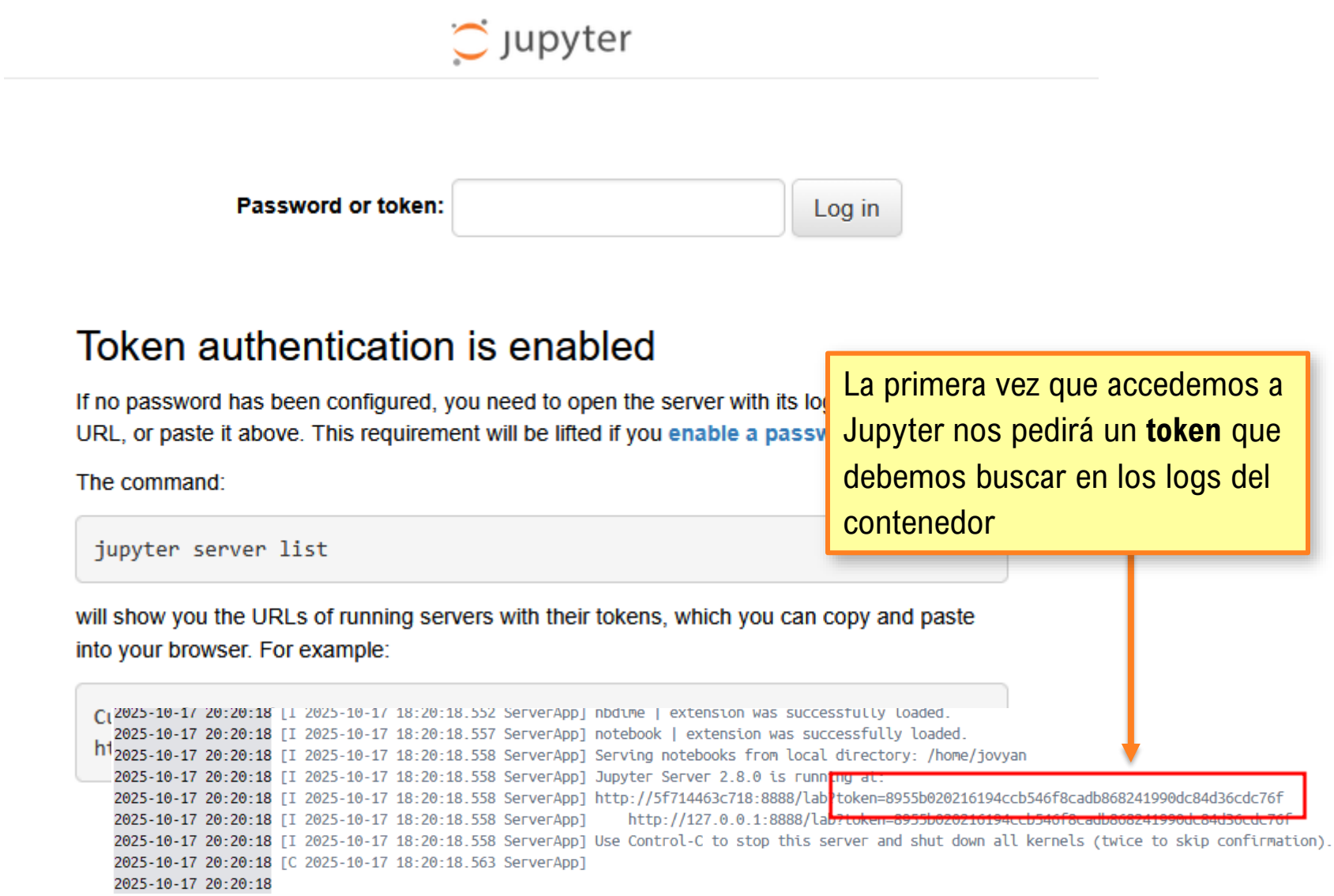
En nuestro caso utilizaremos un **entorno contenerizado** que contiene tanto los nodos de Hadoop, como el servicio con Jupyter.



hadoop	-	-	-	1.34%	47 minutes ago		
namenode	b22f9a357e0f	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1	9000:9000 Show all ports (2)	0.15%	47 minutes ago		
datanode2	f14d746beedd	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-j		0.21%	47 minutes ago		
datanode1	3cfb0ee34cff	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-j		0.23%	47 minutes ago		
resourcemanager	923f913ea2fb	bde2020/hadoop-resourcemanager:2.0.0-hadoo	8088:8088	0.5%	47 minutes ago		
nodemanager1	af8b95063689	bde2020/hadoop-nodemanager:2.0.0-hadoop3.		0.2%	47 minutes ago		
hue	2713ea35e99f	gethue/hue:latest	8889:8888	0.04%	47 minutes ago		
jupyter-notebook	803a554bf2c8	jupyter/pyspark-notebook:latest	8888:8888	0.01%	47 minutes ago		

JUPYTER NOTEBOOK

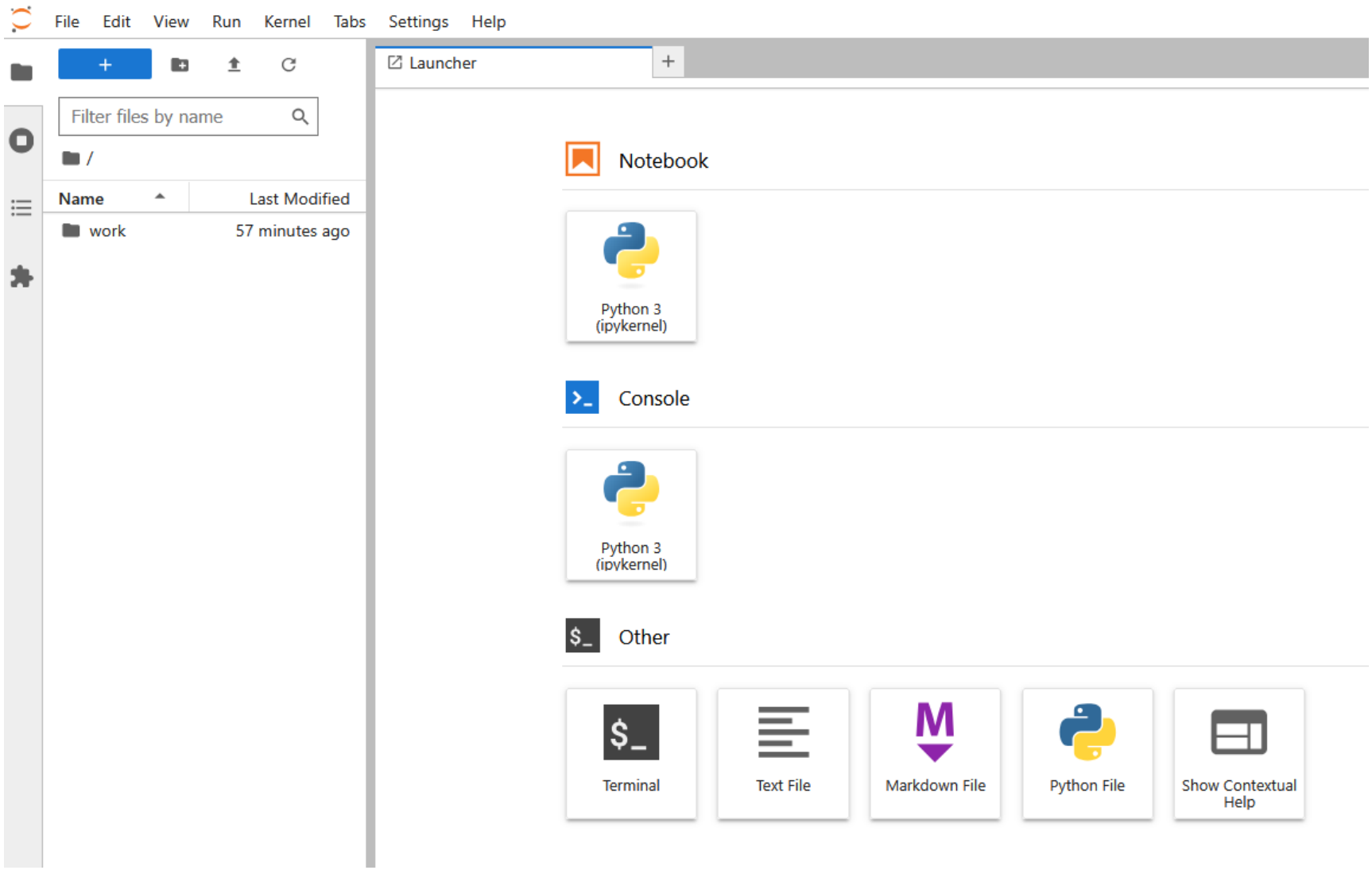
Víctor J. González



La primera vez que accedemos a Jupyter nos pedirá un **token** que debemos buscar en los logs del contenedor

JUPYTER NOTEBOOK

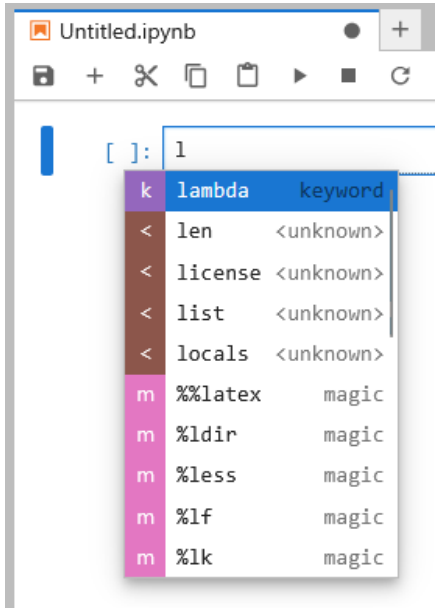
Víctor J. González



Jupyter utiliza **IPython**, que podemos definir como una versión supervitaminada de la consola interactiva de Python.

Algunas características que tiene son:

Autocompletado con Tab: escribes el principio de una variable o función y al presionar Tab mostrará o completará las opciones.



Sintaxis coloreada

```
[ ]: def my_func(a, b):
         return a + b
```

Introspección de objetos: al escribir el nombre de cualquier objeto seguido de un signo de interrogación (?) mostrará su **ayuda**, si son dos interrogaciones (??) se verá el **código fuente**

```
len?
```

```
Signature: len(obj, /)
Docstring: Return the number of items in a container.
Type:      builtin_function_or_method
```

```
my_func??
```

```
Signature: my_func(a, b)
Docstring: <no docstring>
Source:
def my_func(a, b):
    return a + b
File:      /tmp/ipykernel_351/4128038718.py
Type:      function
```

Comandos mágicos: comandos especiales que empiezan con % (comandos de línea) o %% (comando de celda) que no son de Python.

- `%lsmagic`: muestra el listado de comandos mágicos
- `%whos`: lista detallada de todas las variables creadas en la sesión
- `%history`: historial de comandos que se han ejecutado
- `%reset`: borra todas las variables y definiciones creadas
- `%load <file>:` carga el contenido de un fichero de código externo en una celda
- `%time`: mide el tiempo de ejecución de una única instrucción
- `%timeit`: más preciso que `%time` ya que ejecuta el código múltiples veces mostrando el tiempo medio más rápido y fiable
- `%prun`: ejecuta un análisis de rendimiento (*profiling*) de una instrucción, mostrando qué funciones internas tardan más en ejecutarse.
- `%%writefile <file>:` guarda el contenido de la celda entera en un fichero.
- `%cd`: cambia el directorio de trabajo actual
- `%ls`: muestra los ficheros del directorio actual

```
import numpy as np
a = np.random.standard_normal((100, 100))
```

```
a
```

```
array([[-0.07950731, -0.87565537,  0.89871733, ..., -0.3683043 ,
        -0.0698121 ,  0.65124135],
       [-1.74670573, -0.88981375, -0.56607053, ...,  1.74149555,
         1.45707217,  0.94701091],
       [-0.13308989, -2.0108082 ,  0.23944754, ...,  2.11584016,
        -1.73144664,  0.43767848],
       ...,
```

```
%time np.dot(a, a)
```

```
CPU times: user 13.2 ms, sys: 10.4 ms, total: 23.6 ms
Wall time: 4.64 ms
```

```
%timeit np.dot(a, a)
```

```
143 µs ± 47.7 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Además, también podemos ejecutar cualquier **comando del sistema** precediéndolo del símbolo de exclamación (!)

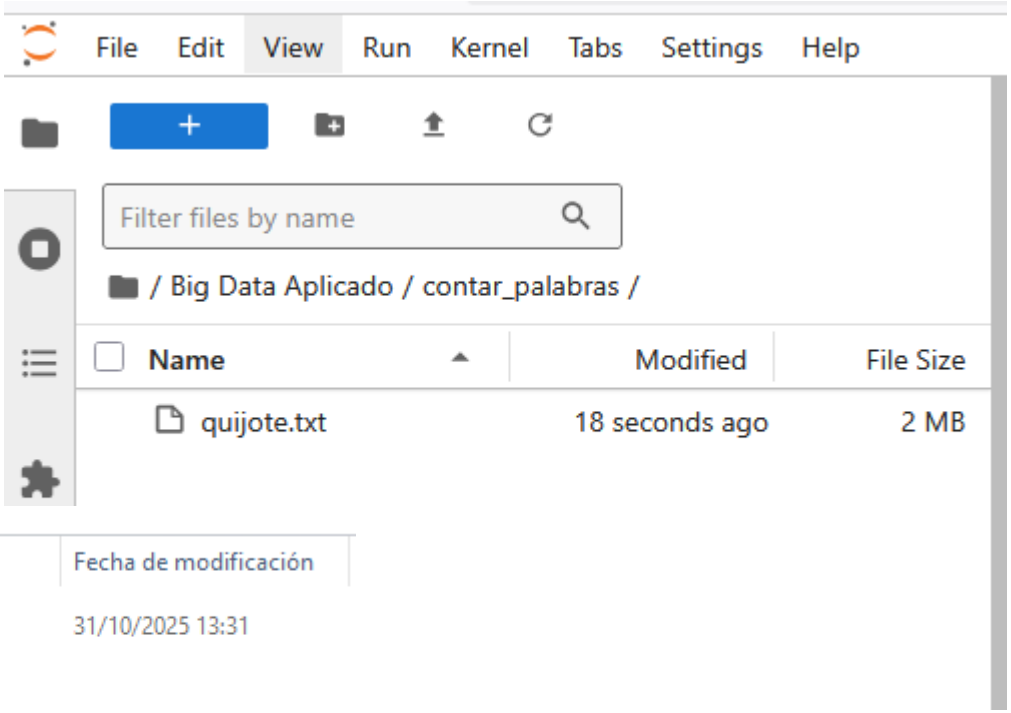
```
!ps
```

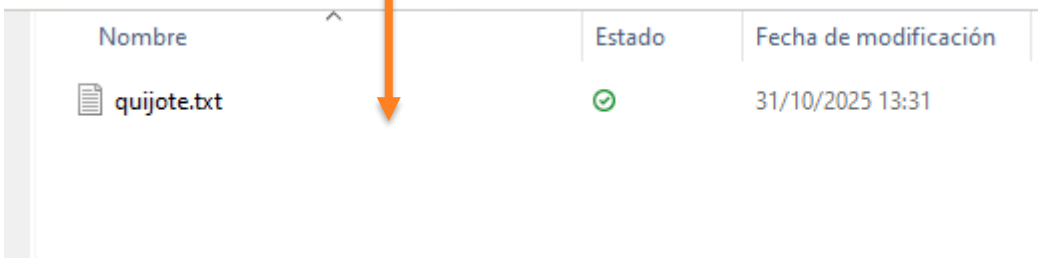
```
  PID TTY          TIME CMD
 1888 pts/0    00:00:00 ps
```

Veamos como lanzar un proceso MapReduce con Python usando Jupyter Notebook con el mismo ejemplo de contar palabras.

```
volumes:
  - ./notebooks:/media/notebooks
```

En el directorio del Docker Compose tendremos un directorio llamado *notebooks*, que está mapeado con un directorio del contenedor. Ahí copiamos nuestros datos.





Comienzo moviendo los datos desde el sistema de ficheros local a HDFS

```
!ls *.txt -l
```

```
-rwxrwxrwx 1 root root 2141519 Oct 31 12:31 quijote.txt
```

```
# Creo un directorio en HDFS
!hdfs dfs -mkdir /contar_palabras
```

```
# Y subo los datos a ese directorio
!hdfs dfs -put ./quijote.txt /contar_palabras/quijote.txt
```

```
!hdfs dfs -ls /contar_palabras
```

```
Found 1 items
-rw-r--r--   3 root supergroup    2141519 2025-10-31 12:38 /contar_palabras/quijote.txt
```

Creamos el fichero con el *mapper*, para ello uso **%%writefile** que almacenará el contenido de la celda en un fichero cuando ejecutemos la celda.

```
%%writefile mapper.py
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

```
Writing mapper.py
```

Y hago lo mismo con el *reducer*

```
%%writefile reducer.py
#!/usr/bin/env python3
import sys

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.strip().split("\t", 1)
    count = int(count)

    if current_word == word:
        current_word += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word:
    print(f"{current_word}\t{current_count}")
```

```
Writing reducer.py
```

Ahora es momento de ejecutarlo sobre el sistema de ficheros local para probarlo

```
!cat "/media/notebooks/Big Data Aplicado/contar_palabras/quijote.txt" \
| python3 mapper.py \
| sort \
| python3 reducer.py
```

```
!Mal    1
"Al     1
"Cuando 2
"Cuidados       1
"De     2
"Desnudo        1
"Dijo   1
"Dime   1
"Don    1
"Donde  1
"Dulcinea       1
"El     2
"Esta   1
"Harto  1
"Iglesia,       1
```

Ahora es momento de ejecutarlo sobre el sistema de ficheros local para probarlo

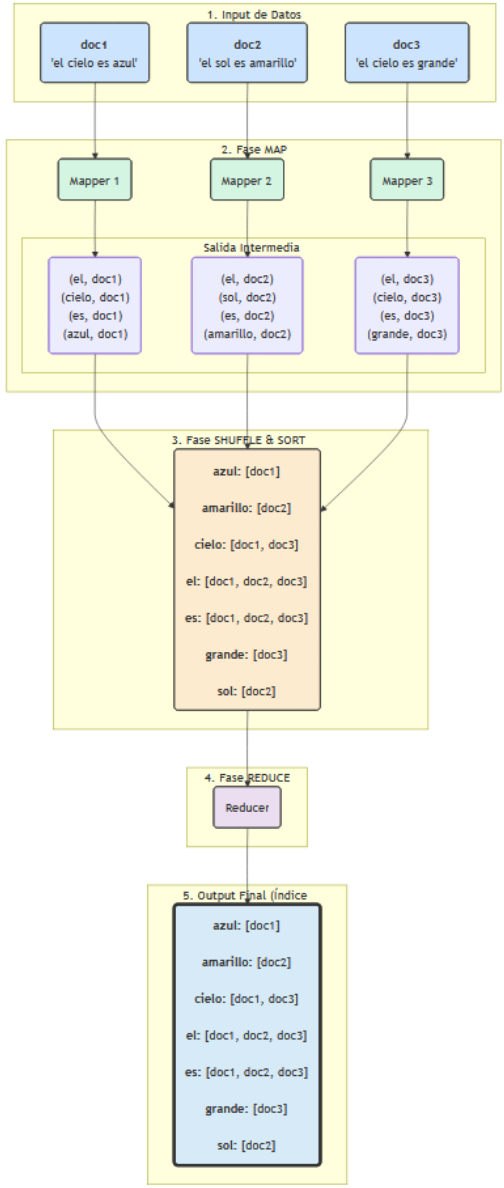
```
!hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.0.jar\
   -input /contar_palabras/quijote.txt \
   -output /salida \
   -file mapper.py \
   -file reducer_2.py \
   -mapper "python3 mapper.py" \
   -reducer "python3 reducer.py"
```

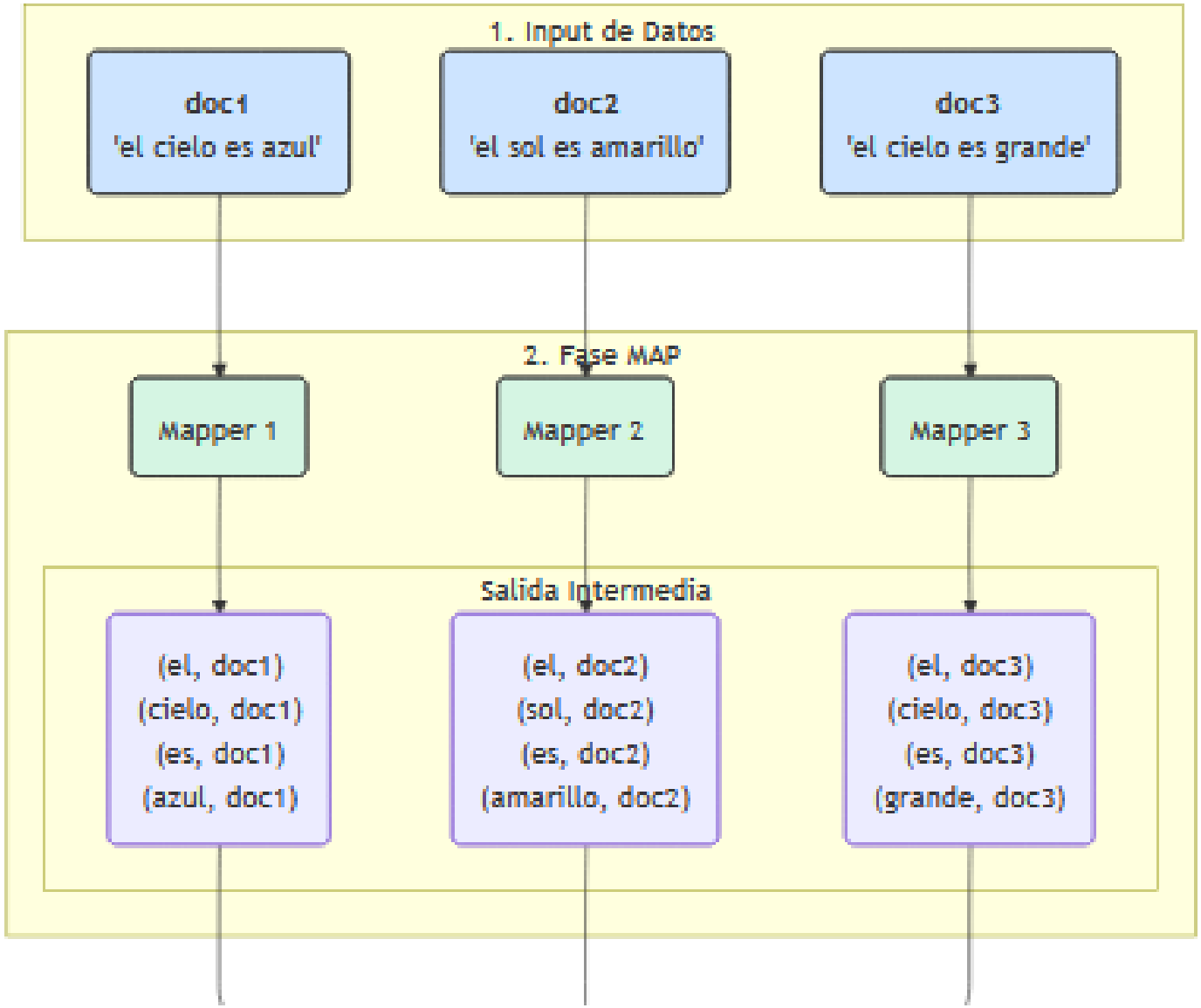
2 MAPREDUCE CON STREAMING PYTHON

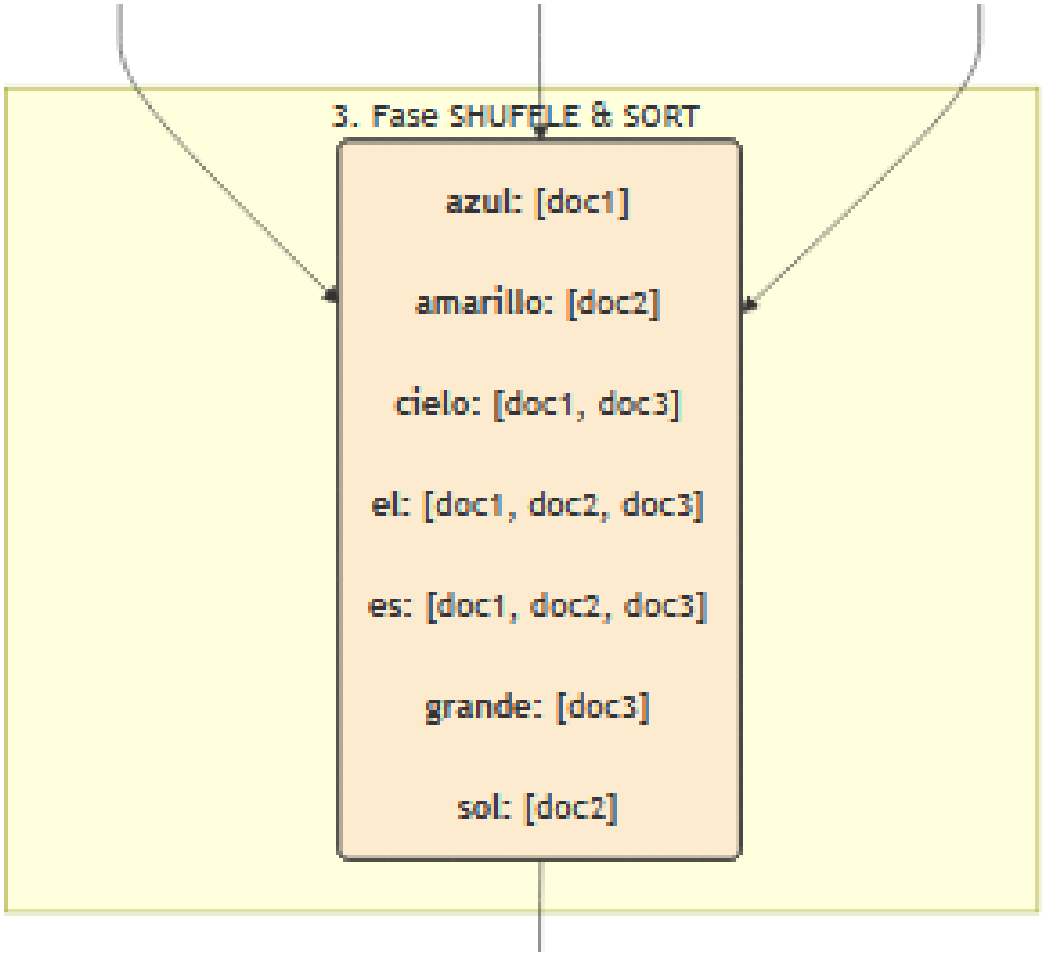
2.3 EJEMPLO ÍNDICE INVERTIDO

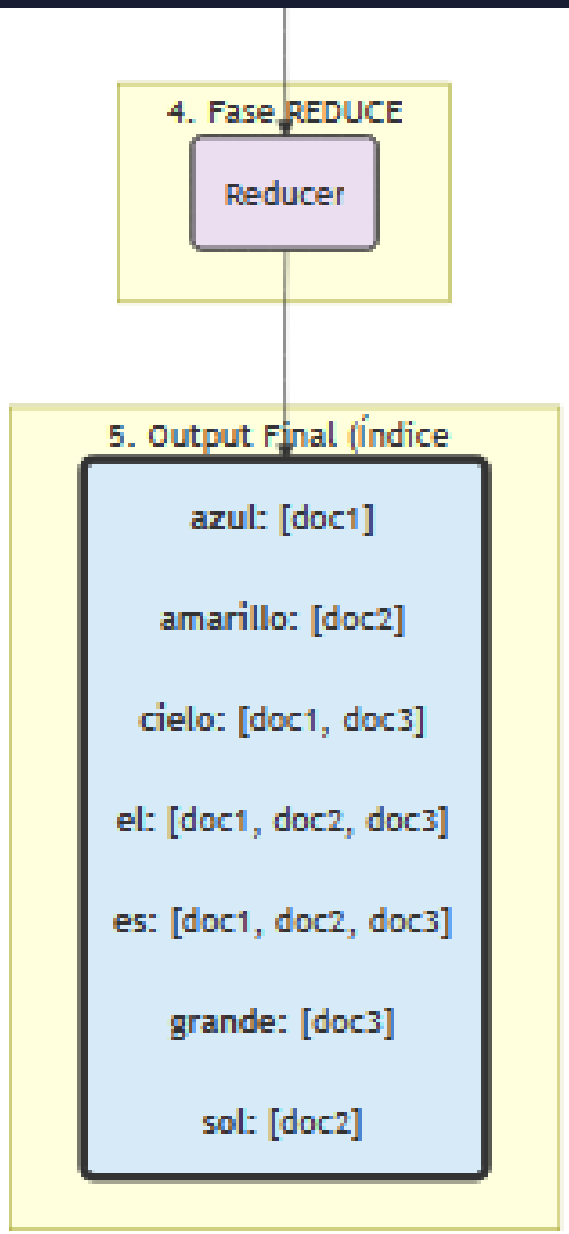
Vamos a ver ahora otro ejemplo que utilice un patrón de diseño diferente: **Inverted Text (Índice Invertido)**, que pertenece a la categoría de **Transformación y Análisis**

Este patrón consiste en construir un índice que **asocia cada palabra con los documentos** en los que aparece. Esto es muy usado en motores de búsqueda (Google, Lucene, Elasticsearch)

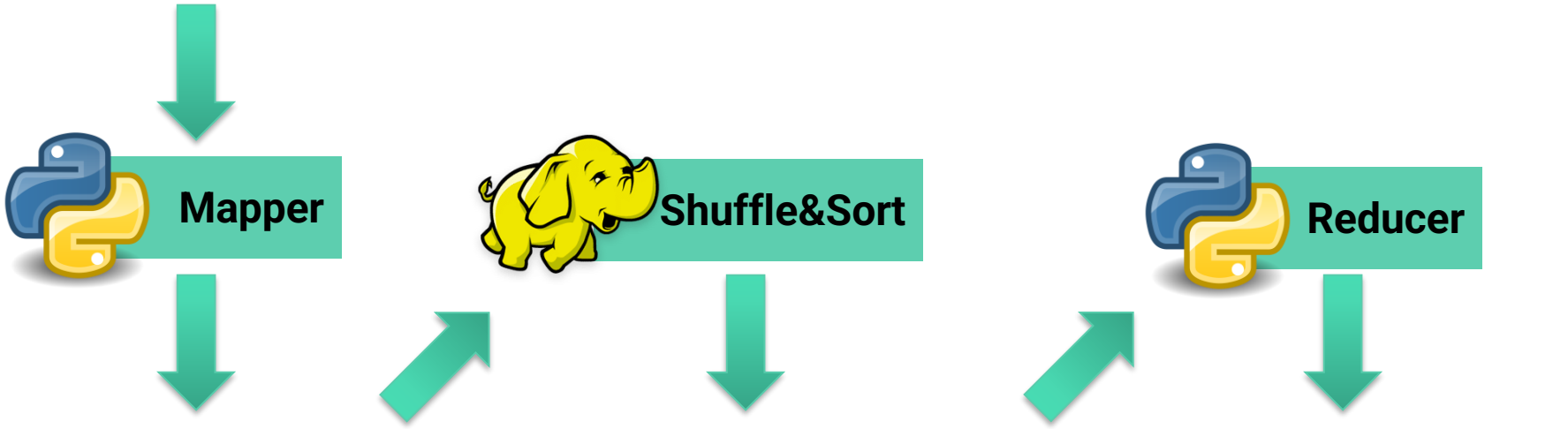



1. Input de Datos
doc1
'el cielo es azul'
doc2
'el sol es amarillo'
doc3
'el cielo es grande'
2. Fase MAP
Mapper 1
Mapper 2
Mapper 3
Salida Intermedia
(el, doc1)
(cielo, doc1)
(es, doc1)
(azul, doc1)
(el, doc2)
(sol, doc2)
(es, doc2)
(amarillo, doc2)
(el, doc3)
(cielo, doc3)
(es, doc3)
(grande, doc3)


3. Fase SHUFFLE & SORT
azul: [doc1]
amarillo: [doc2]
cielo: [doc1, doc3]
el: [doc1, doc2, doc3]
es: [doc1, doc2, doc3]
grande: [doc3]
sol: [doc2]


4. Fase REDUCE
Reducer
5. Output Final (índice
azul: [doc1]
amarillo: [doc2]
cielo: [doc1, doc3]
el: [doc1, doc2, doc3]
es: [doc1, doc2, doc3]
grande: [doc3]
sol: [doc2]

```
doc1.txt
hadoop es rápido y poderoso
doc2.txt
hadoop permite procesamiento distribuido rápido
```




```
hadoop   doc1.txt
es       doc1.txt
rápido   doc1.txt
...
hadoop   doc2.txt
...
```

```
hadoop → [doc1.txt, doc2.txt]
rápido → [doc1.txt, doc2.txt]
...
```

```
es       doc1.txt
hadoop   doc1.txt,doc2.txt
permite  doc2.txt
rápido   doc1.txt,doc2.txt
...
```

Mapper

```python
#!/usr/bin/env python3
import sys
import os

filename = os.environ.get("map_input_file", "unknown")

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word}\t{filename}")
```

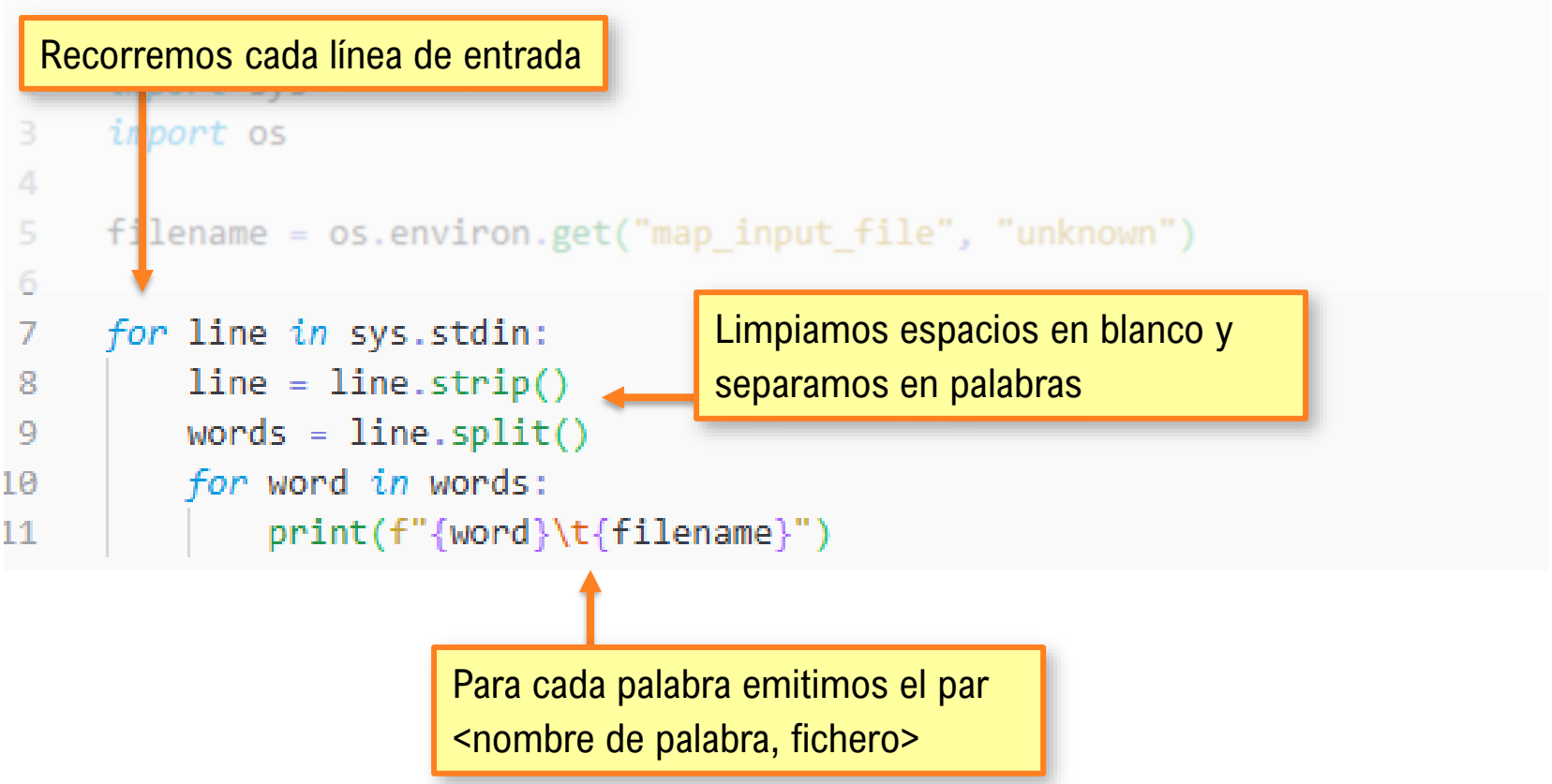
HADOOP STREAMING

Víctor J. González

Por defecto, el *mapper* solo recibe línea a línea el contenido del archivo. Como en este caso necesitamos conocer el **nombre del archivo** debemos usar la variable de entorno `map_input_file` que Hadoop inyecta al script y que contiene la ruta del archivo actual

```
1   #!/usr/bin/env pyth
2   import sys
3   import os
4
5   filename = os.environ.get("map_input_file", "unknown")
6
7   for line in sys.stdin:
8       line = line.strip()
9       words = line.split()
10      for word in words:
11          print(f"{word}\t{filename}")
```

Valor por defecto si no estuviera definida la variable de entorno


Recorremos cada línea de entrada
import os
filename = os.environ.get("map_input_file", "unknown")
for line in sys.stdin:
line = line.strip()
words = line.split()
for word in words:
print(f"{word}\t{filename}")
Limpiamos espacios en blanco y separamos en palabras
Para cada palabra emitimos el par <nombre de palabra, fichero>

Reducer

```python
#!/usr/bin/env python3
import sys

current_word = None
docs = set()

for line in sys.stdin:
    word, doc = line.strip().split("\t", 1)

    if current_word == word:
        docs.add(doc)
    else:
        if current_word:
            print(f"{current_word}\t{','.join(sorted(docs))}")
        current_word = word
        docs = {doc}

if current_word:
    print(f"{current_word}\t{','.join(sorted(docs))}")
```

HADOOP STREAMING

Víctor J. González

```python
#!/usr/bin/env python3
import sys

current_word = None
docs = set()

for l
    w
    i
    e
        if current_word:
            print(f"{current_word}\t{','.join(sorted(docs))}")
        current_word = word
        docs = {doc}

if current_word:
    print(f"{current_word}\t{','.join(sorted(docs))}")
```

Variable para saber cuál es la palabra que estamos procesando

Aquí almacenaremos los ficheros en los que se encuentra la palabra. Utilizamos un conjunto para eliminar automáticamente los duplicados

```python
#!/usr/bin/env python3
import sys

cu
do

for line in sys.stdin:
    word, doc = line.strip().split("\t", 1)

    if current_word == word:
        docs.add(doc)
    else:
        if current_word:
            print(f"{current_word}\t{','.join(sorted(docs))}")
        current_word = word
        docs = {doc}
```

Extraemos la palabra y el nombre del fichero del par enviado por el Shuffle&Sort

Si es la misma palabra que en la iteración anterior, añadimos el nombre del fichero al conjunto

Ordenamos la lista de documentos y los separamos por comas

Si no es la misma palabra emitimos el par con la palabra y el listado de documentos

Pasamos a la siguiente palabra

```python
#!/usr/bin/env python3
import sys

current_word = None
docs = set()

for line in sys.stdin:
    word, doc = line.strip().split("\t", 1)

    if current_word == word:
        docs.add(doc)
    else:
        if current_word:
            print(f"{current_word}\t{','.join(sorted(docs))}")
        current_word = word
        docs = {doc}

if current_word:
    print(f"{current_word}\t{','.join(sorted(docs))}")
```

Y no nos olvidamos de emitir la última palabra

Ejecutamos el programa

```
hadoop@hdfs-server:~/practicas/python_inverted$ hadoop jar \
> /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar \
> -file ./inverted_mapper.py \
> -mapper ./inverted_mapper.py \
> -file ./inverted_reducer.py \
> -reducer ./inverted_reducer.py \
> -input /user/hadoop/input/ \
> -output /user/hadoop/output
```

```
2025-08-31 06:10:18,549 WARN streaming.StreamJob: -file option is deprecated, please use generic
 option -files instead.
```

Si te fijas en la salida nos muestra un warning indicando que la opción `-file` está marcada obsoleta. A partir de la versión 2.x se aconsejable usar `-files` para indicar juntos todos los ficheros separados por comas

Y vemos la salida

```
hadoop@hdfs-server:~/practicas/python_inverted$ hdfs dfs -cat /user/hadoop/output/part-00000
distribuido     hdfs://localhost:8020/user/hadoop/input/file2.txt
es      hdfs://localhost:8020/user/hadoop/input/file1.txt
hadoop  hdfs://localhost:8020/user/hadoop/input/file1.txt,hdfs://localhost:8020/user/hadoop/input/file2.txt
permite hdfs://localhost:8020/user/hadoop/input/file2.txt
poderoso        hdfs://localhost:8020/user/hadoop/input/file1.txt
procesamiento   hdfs://localhost:8020/user/hadoop/input/file2.txt
rápido  hdfs://localhost:8020/user/hadoop/input/file1.txt,hdfs://localhost:8020/user/hadoop/input/file2.txt
y       hdfs://localhost:8020/user/hadoop/input/file1.txt
```

2 MapReduce CON STREAMING PYTHON

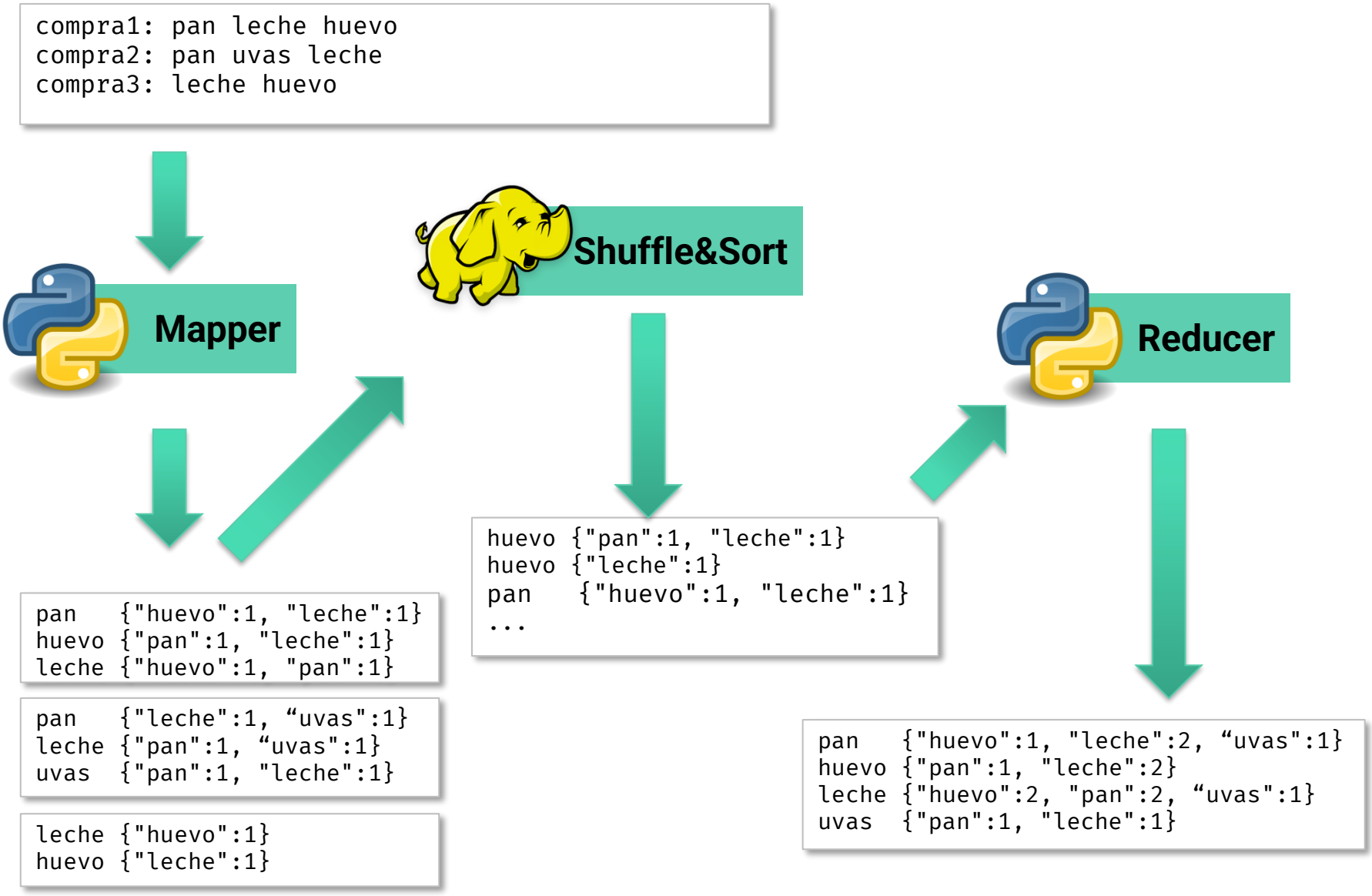
2.4 EJEMPLO: ENCONTRAR CO-OCURRENCIAS DE EVENTOS

Veamos un último ejemplo, en este caso usando el patrón **stripes** cuyo cometido es encontrar **co-ocurrencias de elementos**.

En este caso lo vamos a aplicar a la cesta de la compra y queremos saber **qué productos suelen comprarse juntos en un mismo carrito de compras**.

HADOOP STREAMING

Víctor J. González



HADOOP STREAMING

Víctor J. González

3 PATRONES DE DISEÑO DE MapReduce

PATRONES DE DISEÑO

Víctor J. González

Ahora que ya entendemos cómo funciona una aplicación básica de MapReduce podemos ver qué tipos de problemas son adecuados para este framework.

Para ello tenemos que hablar de **patrones de diseño**, que son plantillas reutilizables que describen cómo estructurar *mappers* y *reducers* para resolver un tipo común de problema de procesamiento de datos distribuidos.

Algunas categorías de patrones de diseño son:

- Patrones de filtrado
- Patrones de agregación
- Patrones de unión
- Patrones de ordenación
- Patrones de validación
- Patrones de transformación
- Patrones de datos complejos

Patrones de FILTRADO

Se usan para **seleccionar** o **descartar** datos

Selection

Filtrar registros que cumplen una condición

Ejemplo: Quedarnos con todas las líneas de un log con error 404)

ESTRUCTURA MAPPER

Lee registros y **emite solo los que cumplen la condición** (clave = registro o campo relevante)

ESTRUCTURA REDUCER

Generalmente **no se necesita**

Patrones de FILTRADO

Se usan para **seleccionar** o **descartar** datos

Projection

Quedarse solo con algunas columnas/atributos de los datos

Ejemplo: Quedarnos solo con la IP de un log

ESTRUCTURA MAPPER

Lee registros y **extrae solo los campos necesarios**

ESTRUCTURA REDUCER

El *reducer* puede ser **identidad** (solo pasa los valores)

Patrones de FILTRADO

Se usan para **seleccionar** o **descartar** datos

Sampling

Tomar una muestra representativa de los datos

Ejemplo: 10% de las líneas de un *dataset* para pruebas

ESTRUCTURA MAPPER

El *mapper* emite registros, pero **filtra aleatoriamente según una probabilidad**

ESTRUCTURA REDUCER

Reducer identidad o ausente.

Patrones de AGREGACIÓN

Sirven para **agrupar y resumir datos**

Word Count

Cuenta las ocurrencias de palabras en un texto

Ejemplo: xxxx

ESTRUCTURA MAPPER

Divide el texto en palabras y emite pares `(palabra, 1)`

ESTRUCTURA REDUCER

Suma todos los valores asociados a cada palabra

Patrones de AGREGACIÓN

Sirven para **agrupar y resumir datos**

Inverted index

Indexar las palabras y en qué documentos aparecen

Ejemplo: Buscadores de Internet

ESTRUCTURA MAPPER

Emite pares (palabra, document_id) para cada palabra en el documento

ESTRUCTURA REDUCER

Agrupa por palabra y devuelve la lista de documentos

Patrones de AGREGACIÓN

Sirven para **agrupar y resumir datos**

Histogramas

Contar cuántos elementos caen en cada rango

Ejemplo: Distribución de edades

ESTRUCTURA MAPPER

Lee valor y emite (rango, 1) según intervalo

ESTRUCTURA REDUCER

Suma las cuentas para cada rango

Patrones de UNIÓN

Permiten **combinar datasets** distintos

Reduce-side join

Juntar datos en el *reduce*

Ejemplo: Unir ventas con datos de clientes

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
Por cada registro de ambas tablas, emite (clave, (origen, valores))	Agrupa por clave y combina registros de ambas tablas

Patrones de UNIÓN

Permiten **combinar datasets** distintos

Map-side join

Si un *dataset* es pequeño, se reparte en todos los nodos y se une en el *mapper*

Ejemplo: xx

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
Mapper accede a un dataset grande y usa en memoria el dataset pequeño (lookup)	Reducer no es necesario (el join ya está hecho)

Patrones de UNIÓN

Permiten **combinar datasets** distintos

Composite join

Combinación optimizada cuando hay varios *datasets* con una clave común

Ejemplo: xxx

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
Emite claves compuestas (clave1, clave2)	Une datos considerando múltiples claves y reorganizando

Patrones de ORDENACIÓN

Relacionados con **ordenar y reorganizar datos**

Total Order Sorting

Ordenar globalmente un *dataset*

Ejemplo: Ordenar logs por timestamp

ESTRUCTURA MAPPER

Mapper emite (clave, valor) normalmente.

ESTRUCTURA REDUCER

Hadoop usa particionadores y comparadores para garantizar el orden global.

Reducer solo escribe

Patrones de ORDENACIÓN

Relacionados con **ordenar y reorganizar datos**

Secondary sort

Ordenar por más de una clave

Ejemplo: Primero por usuario y luego por fecha

ESTRUCTURA MAPPER

Mapper emite clave compuesta (k1, k2)

ESTRUCTURA REDUCER

Reducer recibe datos ordenados por k1 y luego los procesa según k2

Patrones de ORDENACIÓN

Relacionados con **ordenar y reorganizar datos**

Top-N

Obtener los N elementos más grandes/pequeños

Ejemplo: Top 10 palabras más frecuentes

ESTRUCTURA MAPPER

Mapper emite (clave, valor)

ESTRUCTURA REDUCER

Reducer mantiene solo los **N mejores valores** antes de escribir

Patrones de VALIDACIÓN

Su objetivo es verificar la calidad, consistencia o integridad de los datos antes de procesarlos

Validación de formato

Revisa que cada registro cumpla con el formato esperado (número de columnas, tipo de datos, ..)

Ejemplo: Verificar que un dataset de logs tenga 5 campos separados por tabulados

ESTRUCTURA MAPPER

Procesa cada línea, si cumple el formato emite (registro, OK), si no lo cumple (registro, ERROR)

ESTRUCTURA REDUCER

Opcional, agrupa por estado (OK/ERROR) y cuenta cuántos registros de cada tipo hay

Patrones de VALIDACIÓN

Su objetivo es verificar la calidad, consistencia o integridad de los datos antes de procesarlos

Detección de valores nulos o corruptos

Identifica registros con campos vacíos o inconsistentes

Ejemplo: En un dataset de usuarios, comprobar que el campo email no esté vacío

ESTRUCTURA MAPPER

Si el registro tiene valores nulos emite (registro, ERROR), si es válido (registro, OK)

ESTRUCTURA REDUCER

Cuenta el número de registros válidos e inválidos

Patrones de TRANSFORMACIÓN

Modificar, limpiar o enriquecer datos para que sean útiles en fases posteriores

Normalización

Transformar los datos a un formato estándar

Ejemplo: Pasar todos los textos a minúsculas

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
Lee cada registro, aplica la normalización y emite (clave, valor_normalizado)	Opcional, puede combinar resultados de distintas fuentes

Patrones de TRANSFORMACIÓN

Modificar, limpiar o enriquecer datos para que sean útiles en fases posteriores

Parsing

Dividir datos complejos en datos estructurados

Ejemplo: Separar fecha – hora en dos campos distintos: fecha y hora

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
Transforma el campo y emite (nueva_clave, nuevo_valor)	Normalmente no es necesario

Patrones de TRANSFORMACIÓN

Modificar, limpiar o enriquecer datos para que sean útiles en fases posteriores

Enriquecimiento

Añadir información externa o derivada al dataset

Ejemplo: Sustituir el código de país (ES) por el nombre (España)

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
Emite (clave, valor_enriquecido)	Opcional, en caso de conflictos entre varias fuentes, consolida el dato correcto

Patrones de DATOS COMPLEJOS

Extraer conocimiento de los datos mediante cálculos, estadísticas o modelos

Graph Processing

Procesa estructuras de grafos en paralelo

Ejemplo: Algoritmo PageRank

ESTRUCTURA MAPPER

Emite nodos y enlaces (nodo, info)

ESTRUCTURA REDUCER

Combina información de vecinos y actualiza el valor del nodo

Patrones de DATOS COMPLEJOS

Extraer conocimiento de los datos mediante cálculos, estadísticas o modelos

Matrix Multiplication

Multiplica matrices distribuidas

Ejemplo: Entrenamiento de machine learning

ESTRUCTURA MAPPER	ESTRUCTURA REDUCER
XX	XX

Patrones de DATOS COMPLEJOS

Extraer conocimiento de los datos mediante cálculos, estadísticas o modelos

Stripes

Calcula co-ocurrencias de elementos

Ejemplo: Palabras que aparecen juntas o productos que se compran en conjunto

ESTRUCTURA MAPPER

Mapper emite (palabra, {co-palabras}) como diccionario parcial

ESTRUCTURA REDUCER

Reducer fusiona diccionarios y suma frecuencias